0-315-01240-4

National Library of Canada | Bibliothèque nationale du Canada

Canadian Theses Division | Division des thèses canadiennes

Ottawa, Canada K1A 0N4

49098

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

JOHN JOSEPH GILBERT SAVARD

Date of Birth — Date de naissance

8 NOVEMBER 1955

Country of Birth — Lieu de naissance

CANADA

Permanent Address — Résidence fixe

#75, 11255 - 31 AVE EDMONTON, ALBERTA

T6J 3V6

Title of Thesis — Titre de la thèse

THE AUTOMATIC CONTROL OF A 7.5 MEV VAN DE GRAAFF ACCELERATOR

University — Université

UNIVERSITY OF ALBERTA

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

MASTER OF SCIENCE

Year this degree conferred — Année d'obtention de ce grade

FALL, 1980

Name of Supervisor — Nom du directeur de thèse

DR. W. KENNETH DAWSON

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date

OCTOBER 17, 1980

Signature

John Savard

NL-91 (4/77)

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Ottawa, Canada
K1A 0N4

NL-339 (Rev. 8/80)

THE UNIVERSITY OF ALBERTA

THE AUTOMATIC CONTROL OF A 7.5 MEV VAN DE GRAAFF ACCELERATOR SYSTEM

by

© JOHN J. G. SAVARD

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

Physics

EDMONTON, ALBERTA

FALL 1980

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled THE AUTOMATIC CONTROL OF A 7.5 MEV VAN DE GRAAFF ACCELERATOR SYSTEM submitted by John J. G. Savard in partial fulfilment of the requirements for the degree of Master of Science.

.....

Supervisor

.....

.....

.....

Date. October 15/80.....

Abstract

An attempt to monitor and control a 7.5 MeV Van de Graaff and associated equipment by means of a minicomputer through the use of CAMAC modules for interfacing is described. The accelerator system itself, the requirements of machine operators, and the programming systems used are described. The existing control hardware configuration is described, with some discussion of the consequences of possible future extensions to it.

Acknowlegments

I would like to acknowlege the advice and encouragement of my supervisor, Dr. W. K. Dawson, the assistance in understanding the Van de Graaff accelerator provided to me by Dr. Tsing-Lan Lam , Dr. Harold Fielding, and Mr. J. B. Elliott, and the assistance and co-operation of the past and present members of the Nuclear Research Center's electronics shop, including but not limited to Mr. Lars Holm, Mr. John Schaapman, Mr. R. Popik, Mr. John Ritzel, Mr. D. Presakarchuk and Mr. John Hewlett.

Also, for his advice concerning the use of the University of Alberta TEXTFORM system, I must thank Mr. David Holberton of Computing Services.

Table of Contents

Chapter		Page
I	INTRODUCTION	1
II	ACCELERATOR SYSTEM DESCRIPTION	2
	2.1 Introduction	2
	2.2 The Van de Graaff	2
	2.3 The Top End	5
	2.4 The Beam Line	8
	2.5 The Slit Feedback System	11
	2.6 The Mobley Beam Compression System	14
III	CAMAC	17
	3.1 Introduction	17
	3.2 The Crate	18
	3.3 Multi-Crate Systems	20
IV	BASIC	21
	4.1 Introduction	21
	4.2 Modifications to BASIC	22
V	FORTH	24
	5.1 Introduction	24
	5.2 Basic Concepts	25
	5.3 Language Description	26
	5.4 The Dictionary	32
VI	CONTROL REQUIREMENTS	35
	6.1 Introduction	35
	6.2 Operator Services	35
	6.3 Accelerator Control	37

VII HARDWARE CONFIGURATION 42
7.1 Introduction 42
7.2 The Computer System 43
7.3 The CAMAC System 43
7.4 Console Electronics 44
7.5 CAMAC Modules 45
VIII OPERATOR INTERFACE 49
8.1 Introduction 49
8.2 Computer Outputs 49
8.3 Computer Inputs 53
IX PROGRAM ORGANIZATION 55
9.1 Introduction 55
9.2 The Console Scanner 56
9.3 Accelerator Control 57
X CONCLUSIONS 66
10.1 Introduction 66
10.2 Programming Languages 66
10.3 Operational Reliability 68
10.4 Hardware Extensions 70
Bibliography 77
Appendix I
FORTH Control System Source Code 79

List of Figures

Figure		Page
1.	The Mobley Beam Compression System . . .	16
2.	The Accelerator Control Hardware System	43

CHAPTER I

INTRODUCTION

This thesis describes a series of preliminary program development efforts aimed at computer control of the 7.5 MeV Van de Graaff accelerator and associated beam lines located at the University of Alberta Nuclear Research Center.

Introductory chapters describe the principles of accelerator system operation, the existing hardware facilities for machine control, and programming languages used. As well, the modifications made to one programming language system (BASIC) to transform it into a more effective instrument of machine control are described.

After discussing the requirements of operator service and machine control, the organization of the present version of the console scanner/control program is described relative to these requirements.

The present control program contains the framework for machine control, but actual machine control has not yet been implemented.

CHAPTER II

ACCELERATOR SYSTEM DESCRIPTION

2.1 Introduction

The purpose of the accelerator system is to deliver a beam of energetic charged particles to an experimental target. Thus, a beam of the desired species of charged particles must be produced, accelerated to the required energy, and the beam must be directed towards the target, and focused upon it.

The Van de Graaff proper is a device to create a high positive potential on a metal shell which surmounts it. It is within that shell that the charged particles are produced, and it is the potential there that provides the particles with their energy.

The charged particle beam is kept within evacuated tubing which also encloses the target. On its way, it is bent and focused by a number of magnetic and electrostatic lenses. The beam may be pulsed so that the time between when particles strike the target and particles or gamma rays leaving the target are detected may be observed in experiments.

2.2 The Van de Graaff

A Van de Graaff accelerator consists fundamentally of a metal shell, supported by an insulating column with a charged insulating belt moving within.

A high-voltage discharge places positive charge on the belt at the base of the column. As the belt is insulating, charge cannot flow along it and therefore must follow its motion against the potential gradient inside the column, gaining electrostatic potential energy as it moves. Once inside the metal shell atop the column, the charge on the belt has gained enough energy to be at a potential comparable to that of its new surroundings. Thus, as though the charge had not travelled across a potential difference of several million volts, it may still freely leave the belt and add to the charge collected on the outer surface of the shell. As well, negative charge is sprayed on the belt by another high-voltage discharge before the belt leaves the metal shell on its way downwards.

It is intended to maintain a potential of up to seven million volts on the metal shell atop the Van de Graaff. Therefore, it is necessary to take precautions to minimize the loss of charge from the shell, and to prevent destructive sparks.

One of the precautions taken is that the entire Van de Graaff is placed within a large vesel filled with compressed insulating gas when in operation. Another is that the potential gradient along the column is kept approximately uniform, to prevent large local potential differences from resulting in sparks.

For this purpose, the entire column is ringed by a set of metal bands. Each of these bands is connected to the one

above it and the one below it by a large resistor (500-1200 megohms), the topmost ring being connected to the charged shell atop the column, and the lowest ring being connected, through an ammeter, to ground. Also present is a series of smaller rings surrounding the belt (insulating spacers prevent the belt from touching these rings), and a series of thin metal plates sandwiched between the rings of insulating material which form that section of evacuated beam tube which is within the column. The items in these two series have the same vertical spacing as the metal bands around the outside of the column, and each ring around the belt and each metal plate in the beam tube is connected to the metal band around the outside of the column at the same height, in order to more fully maintain a smooth potential gradient within the column.

The chain of resistors used to maintain a smooth potential gradient along the column is one source of loss of charge from the metal shell atop the column. Corona discharge from the shell is another source of charge loss in normal operation; the current lost in each of these pathways is from 10 to 50 microamperes.

The rate of charge loss through corona discharge is controlled by the corona points. These are negatively charged needles which protrude from a metal screen attached to the containment vessel. The corona points can be moved towards or away from the charged metal shell, and their charge is adjusted automatically by the slit control system,

which will be described below.

Also on the inside of the containment vessel is the generating voltmeter. It consists of a rotating flat fanlike piece of metal, in front of a set of stationary 'blades' which are alternately shielded and exposed by the rotating element in front. Thus, the stationary blades in the generating voltmeter are connected to the top end, which is the metal shell atop the accelerator and the equipment contained within the shell, by an oscillating capacitance. Thus, if the stationary blades are connected to ground through a resistor, an AC current proportional to the top end voltage will flow.

2.3 The Top End

By placing the top end at a high positive voltage, it is ensured that a beam of positively charged particles originating there will have a high energy. Therefore, the source of ions and some equipment for focusing the beam are placed within the dome-shaped metal shell atop the Van de Graaff accelerator.

The ion source, like the entire beam pipe, is evacuated to permit the free movement of charged particles within. A small amount of the gas the ions of which are to be accelerated is allowed to seep into the ion source, through a slow leak whose permeability is controllable.

Radiofrequency energy, at a frequency of 125 MHz, heats and ionizes the gas in the ion source. An electromagnet, the

source magnet, around the ion source concentrates the ionized gas, making the process more effective. The level of the 125 MHz RF energy cannot be adjusted during operation: therefore, it is the source magnet current that controls the rate of ion production. If the source magnet current is too high, the plasma will draw too much current from the RF oscillator; the recommended maximum level for the oscillator plate current is 350 mA.

The ion source itself is a glass tube, ringed by two metal bands to supply RF energy to the plasma within, and surmounted by a metal cap. A metal nozzle protrudes into the ion source at its base through a small hole in a flat metal plate, the source aperture plate, where the ion source opens into the beam pipe. The voltage difference between the metal cap atop the ion source and the nozzle below is called the probe voltage, and it controls the rate at which ions are pulled from the ion source.

The top element of a three-element cylindrical electrostatic lens tapers at its top to become that nozzle. The voltage on the middle electrode is controlled to alter the focusing effect of the lens, while the other two elements are electrically connected. This lens, called an Einzel lens, is the first focusing element in the system and is called Focus 1.

Relative to the potential at the source aperture plate, the metal cap atop the ion source, called the probe, is at a voltage of up to +15 kV, the middle electrode in the Einzel

lens is at a negative voltage of up to -10 kV, and the other two electrodes are at a negative voltage of up to -30 kV.

In the particular accelerator system under discussion, there are two assemblies of ion source and Einzel lens such as the one described above. The elements of the two Einzel lenses are electrically linked; the incoming flow of gas, source magnet current, and source oscillator current are connected to the source in use.

Which source is in use is determined by the direction of the current in the terminal analysis magnet, the next place the beam enters. As the terminal analysis magnet bends the beam magnetically, and the force on a charged particle in a magnetic field depends on its velocity as well as its charge, most impurities are removed from the beam on the basis of their charge-mass ratio. Even at this point, the kinetic energy of each particle is proportional to its charge, as the particles have travelled through a potential difference of several thousand volts.

Next, the beam passes through two sets of parallel electrostatic deflection plates, each set at 90° to the other. An RF signal is applied to the plates, with that applied to one set out of phase with that applied to the other set by 90°, deflecting the beam in a constantly changing direction. Below the deflection plates, there is a flat metal plate in the path of the beam with a small hole in the center. On this plate, the beam traces a distorted elliptical path when the deflection plates are in use: a DC

bias is used to cause this path to traverse the hole in the chopping aperture plate.

It is necesary to deflect the beam in two dimensions so that if it goes through the hole in the chopping aperture plate only once, not twice, per RF cycle. This way, the time from one pulse to the next naturally tends to be uniform, rather than having to be made uniform by a critical and unstable adjustment of the deflection plate bias.

Below the chopping aperture plate, there is a conical electrode used to focus the beam. This electrode is called Focus 3, and it is at a negative voltage of up to -40 kV relative to the chopping aperture plate, which is at the same potential as the two linked electrodes in the Einzel lens above. Whether or not a pulsed beam is in use, Focus 1 is set to focus the beam to a point at the hole in the chopping aperture plate.

Focus 3 stands at the entrance to the insulating column, and thus at the exit from the top end.

2.4 The Beam Line

The potential gradient down the insulating column, while smooth, is not precisely uniform: this enables the column itself to have a focusing effect on the beam. As the potential difference along the column is much greater than the differences within the top end, no further controls that can be adjusted while the machine is in use are found after the beam leaves the top end until the beam exits from the

base of the column.

The analyzing magnet, which bends the beam from the accelerator by 90° sits directly below the column, on the floor of the room underneath the Van de Graaff. This magnet removes impurities from the beam in the same fashion as the terminal analysis magnet, and is even more effective at doing so. The analyzing magnet also lets the Van de Graaff stand vertically and be supported by a single insulating column designed primarily for compressional strength, while the experimental area receives a horizontal beam, as is required for convenience in mounting and moving experimental equipment.

The primary purpose of the analyzing magnet is to facilitate the measurement of the energy of the particles in the beam, thus permitting this energy to be controlled. Since the angle by which a charged particle is deflected by a magnetic field depends on the velocity, and hence the kinetic energy, of the particle, for any value of the analyzing magnet current there is only one energy at which a beam of any one particle type will be deflected by a right angle.

The analyzing magnet also acts as a converging lens, and is designed to have its minimum aberration for one specific set of object and image points. In the beam line above the analyzing magnet, there is a small ring surrounding the object point. Focus 3 is to be adjusted so that the beam is focused to a sharp point there: the current

striking the ring, called the focus current, indicates when this goal is not achieved. At the image point to the east of the analyzing magnet, there is a horizontal slit bounded by two adjustable metal wedges. Particles with too low an energy are deflected through an angle greater than 90°, and strike the upper wedge, while particles with too high an energy strike the lower wedge. The amount of current striking each wedge is used by the slit feedback system to determine if the accelerator is actually producing particles at the intended energy. The slit feedback system will be described in detail below.

Also above the analyzing magnet, but below the column, there is a set of steering magnets. Each steering magnet is a specially-wound magnetic deflection yoke, which can deflect the beam by a limited amount in any direction perpendicular to its axis. Two steering magnets separated by a short distance along the beam can together also displace the beam in a direction perpendicular to its own axis. Another set of steering magnets is located between the analyzing magnet and the switching magnet, the next large electromagnet to be encountered.

Within the switching magnet, the evacuated tubing in which the beam travels branches off into three directions. The level and polarity of current in the switching magnet determines the beam line down which the beam will travel. In two of the three beam lines, the beam is focused by quadrupole magnets.

While an electrostatic lens is straightforward in principle, less elaborate equipment and less electrical power is required to deflect a highly energetic charged particle beam to any given extent by the use of electromagnets. However, a simple converging lens cannot be constructed directly with magnetic fields.

A quadrupole magnetic field can be produced that is converging in one plane and diverging in the other plane: two such fields, each rotated by 90° with respect to the other and separated by a distance along the beam, so that second-order effects can be produced by opposite and nearly equal components of their fields, can produce a net focusing effect in both planes. Since the focusing effect is obtained by a different sequence of lenses in each plane, astigmatism is present although the focal length may be the same in each plane. While astigmatism can be reduced by using three quadrupoles, the one in the middle being twice as large as the outer two, this is not done in the system under discussion: instead, the astigmatism of the quadrupoles is used to correct for astigmatism in other parts of the system, primarily the analyzing magnet and the switching magnet.

2.5 The Slit Feedback System

The voltage on the top end of the accelerator is determined by the amount of charge stored there, as the top end acts as one plate of a capacitor to ground. The rate of

charge loss by corona discharge and through the column resistors increases with top end voltage, while the rate of charge gain through the belt, constant unless altered by the operator, is largely independent of top end voltage. Thus, for any given level of belt charge, there is a voltage at the top end that will produce equilibrium. The magnitude of beam current is negligible compared to that of the other charge flows in the accelerator.

While the belt charge must be at a level appropriate to the energy being sought, changes to the belt charge take a considerable time to be reflected in the top end voltage. The mechanical motion of the charged belt imposes a delay on the order of a tenth of a second but an even longer time is needed for equilibrium to be reached after a significant change in the level of charge. The voltage on the corona points is used for fast, delicate control of top end voltage, as here there is a direct electrical link to the charged metal shell atop the accelerator, thus permitting control at electronic speeds. Although the time needed to reach an equilibrium follows the same proportional relation to the amount of voltage change desired whether the belt charge or the corona points are used, since other delays involved are very small, minute changes in the top end voltage can be effected in milliseconds or less.

The voltage on the corona points is controlled by the slit feedback system. The slit feedback system senses the energy of the machine in two ways: through the energy slits

located at the exit focus of the analyzing magnet, and from the generating voltmeter.

A magnetometer utilizing nuclear magnetic resonance is used to measure the field within the analyzing magnet, and the analyzing magnet current is set for the exact energy required with the aid of the NMR magnetometer. Thus, current hitting either one of the wedges at the energy slits predominantly indicates an error not in the analyzing magnet setting, but in the energy of the particles actually being produced by the accelerator.

While the generating voltmeter provides a far less precise indication of top end voltage than the energy slits, this indication is always present, while the energy slits are only useful if a beam close to the desired energy is actually going all the way down to the analyzing magnet and beyond. Thus, the two methods of determining top end voltage complement each other. A three-position switch on the slit feedback system enables the operator to select which of these two methods is used, or to allow the slit feedback system, in "conditional" mode, to perform this selection itself.

In conditional mode, the slits are normally used, but if there is no beam going through the slits, or if the generating voltmeter indicates a voltage error of over 30 kV, the generating voltmeter is used instead. The latter criterion is to prevent the system from locking on to a spurious beam of particles with a charge-to-mass ratio very

close to that of the particles in the intended beam, an applicable example of which is doubly-charged Helium-4 and singly ionized deuterium.

The sensitivity of the feedback loop provided by the slit feedback system is changed by moving the corona points. The sensitivity must be enough to provide tight control, but not enough to cause oscillation, and it must be reduced by retracting the corona points when the energy is raised.

2.6 The Mobley Beam Compression System

One of the three beam lines after the switching magnet contains a system to take a pulsed beam produced by chopping it in the top end and compress each pulse from a duration of about 10 nanoseconds to a duration on the order of one nanosecond. This system, the Mobley beam compression system, has as its most visible element the Mobley magnet. This magnet, like the analyzing magnet, is a double focusing magnet which deflects the beam by 90°, but in this case the beam is in the horizontal plane both upon entering and emerging. The target is located at the image point of this magnet, and the set of quadrupoles present in this beam line is used to focus the beam at the object point of the Mobley magnet.

Particles leaving the object point of the magnet within a limited angular range all go through the Mobley magnet and arrive at the target: but they do not all take the same amount of time to get there. It is this property of the

Mobley magnet that is exploited to obtain beam compression.

A set of electrostatic deflection plates, supplied with an alternating current, located at the object point of the Mobley magnet continuously changes the path by which an incoming beam will go through the Mobley magnet. A capacitative pickoff located at an earlier point in the beam line detects each pulse in the original beam as it goes by. The signal from the capacitative pickoff is used to lock the phase and frequency of the alternating current on the Mobley deflection plates to that of the beam pulses.

This is used to ensure that the earlier part of a beam pulse takes a longer path to the target than the later part, making all parts of the pulse arrive at the target at more nearly the same time. This principle is illustrated by Figure 1.

The operator can adjust both the phase and the amplitude of the signal applied to the deflection plates: the phase is often adjusted to obtain the best quality of pulse, while the correct amplitude setting can be determined from a formula using the particle mass, charge, and energy.

The finite width of the beam, among other things, limits the effectiveness of this method: on the other hand, placing a lens that is diverging in the horizontal plane between the Mobley magnet and target can deal with the penalty of decreased angular resolution that this method appears to impose.

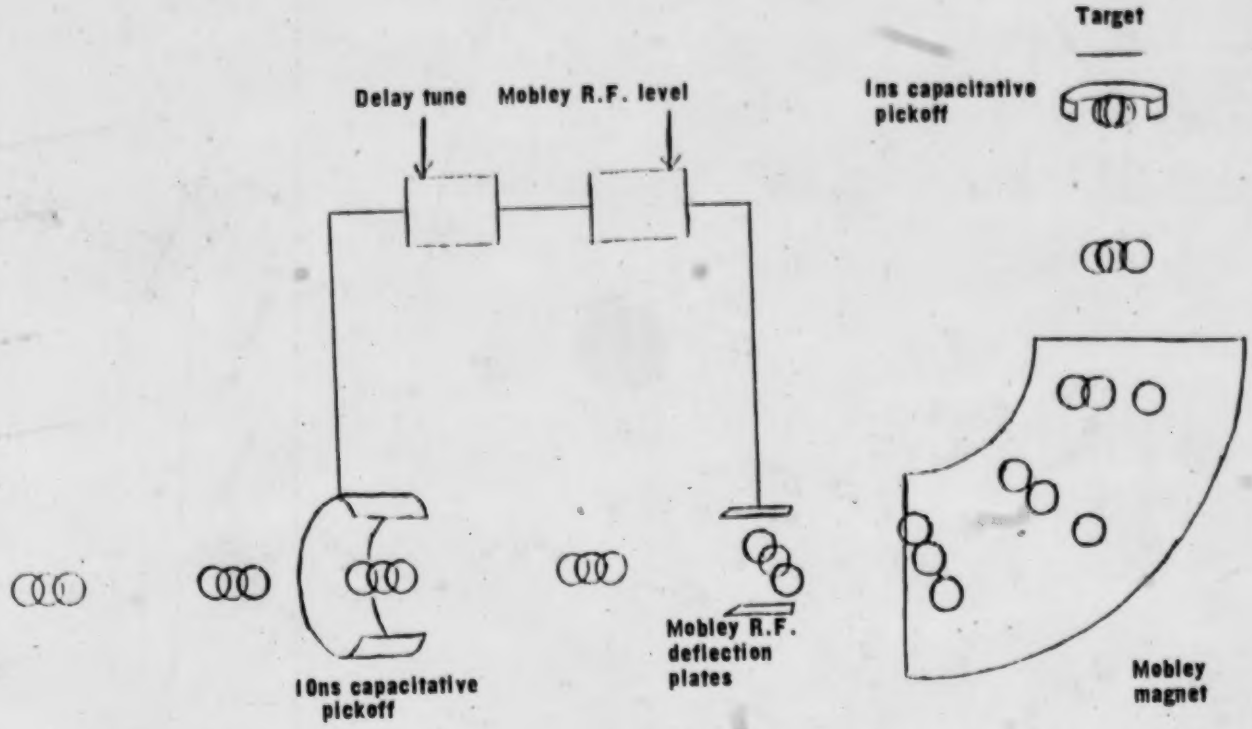

Figure 1: Mobley Beam Compression System

CHAPTER III

CAMAC

3.1 Introduction

CAMAC (Computer Automated Measurement And Control) is a standard defining the digital bus, including power supply ratings and fan-in/fan-out specifications for digital signal sinks/sources, and the physical configuration for mounting the digital modules that conform to the CAMAC standard.

Modules used in CAMAC systems vary in function and complexity from scalers, A/D converters, and I/O ports to waveform digitizers and video display controllers. CAMAC is used in process control, research, and medicine, among other fields.

Because the digital bus defined by CAMAC is processor-independent, the wide variety of digital modules available for CAMAC is available to users of any digital computer for which a CAMAC-to-computer interface can be obtained. Data is normally transmitted over this bus in binary or packed-decimal format, whereas in IEEE standard 488-1975, also known as the GPIB bus, data is transmitted in the form of ASCII character strings to free-standing programmable instruments. Neither it nor any other standard directly competes with CAMAC.

3.2 The Crate

The CAMAC crate is the standard rack in which CAMAC modules are located when in operation. The crate is also a unit of the CAMAC addressing scheme. Up to 25 positions for modules, called stations, may be present in a CAMAC crate. The rightmost station is called the control station, and is not wired in the same way as the other stations, called normal stations. The control station must be one of the stations occupied by the crate controller, which must also occupy at least one other station.

The crate controller decodes five bits of address information, the station number, which identify the module involved in a transaction, and then sends a signal from the control station, called n, to the normal station that is addressed. An additional four bits of address information, called the subaddress, are received directly by the modules, which decode this information themselves so that multiple capabilities within a single module can be accessed.

A five-bit function code identifies the type of transaction taking place. in addition to distinguishing between read, write, and dataless (control) cycles, it describes the purpose of the transaction to a limited extent. For example, F(0) indicates that a data register is being read, while F(1) indicates that a control register is being read, and F(2) indicates that the same data register that F(0) would refer to under the same circumstances is being read and cleared. F(8) tests if the module is

attempting to request service with a Look-At-Me signal, and F(10) clears such a request. F(24) disables some function of the module, and F(26) disables the same feature (if issued to the same module with the same subaddress). Defining the function codes in detail in this way, rather than adding a few bits to the subaddress, enhances compatibility between modules, and causes the software even for completely different types of modules to include common and recognizable elements, thus making software for CAMAC modules understandable to some extent even to those unfamiliar with the specific modules used. The subaddress indicates which of a number of actions is performed by a module where there are several possible actions of the same general type, thus associated with the same function code, available from that module.

Data transmission is along two sets of lines, one for reading and one for writing, each one 24 bits wide. Several signals are common to all stations, among them the Z signal, which initializes all modules, and the Q signal, which is available to whichever module is addressed to send a single bit of status information to the computer during any CAMAC cycle, whether read, write, or control. Each station has a single line direct to the control station which can be used by the module in that normal station to send a request for service to the crate controller. These lines, called the L or Look-At-Me lines, require half of those contact positions in the control station that are used for read and write

lines in normal stations. The same is true of the N lines, which is why the crate controller must occupy a normal station in addition to the control station.

3.3 Multi-Crate Systems

Many CAMAC systems consist of more than one crate. There are two standard methods of connecting crates in a system, the parallel highway and the serial highway. The serial highway was originally devised for systems in which crates are separated from each other by long distances and which could tolerate a slower rate of intercrate communication.

The parallel branch highway is the method used in the system under discussion. In it, up to seven crates may be connected in a branch. Each crate in the branch must have a CAMAC Type A crate controller, or near equivalent, to which connection is made. The nature of the branch driver, which controls a whole branch, is not defined in any CAMAC standard, thus in this installation a system called the Elliot Executive Suite is used, wherein the computer controls one crate, belonging to no branch, which contains the branch drivers for up to four branches as well as modules that can be used normally.

CHAPTER IV

BASIC

4.1 Introduction

BASIC (Beginners' All-purpose Symbolic Instruction Code) is a programming language originally developed by J.G. Kemeny and T. E. Kurtz at Dartmouth College in 1965.

Most implementations of BASIC are by means of an interpreter, rather than a compiler. Two general ways exist in which a computer can be made to automatically perform a computation described in a language other than its own machine language. Either the program can be converted to an equivalent program in the computer's own machine language by a compiler, or a program called an interpreter can read the program describing the computation to be performed, performing the computations that that program calls for while reading it. If this method is chosen, it is considerably less difficult to provide the user with a conversational environment where programs can be modified, run, and modified again, either after normal program exit or after interrupting the program, without having to unload the user program in order to edit and compile its source text again. However, higher-level language programs run considerably more slowly when interpreted than when compiled.

The BASIC language is similar to FORTRAN, and is useful for the same type of work, mathematical calculations in

science and engineering. However, many of the more difficult to understand features of FORTRAN have been omitted or replaced by something easier to use in BASIC.

4.2 Modifications to BASIC

As BASIC possesses the dual advantages of ease of learning and ease of use, as well as because it has been implemented on a wide variety of minicomputers, a standard[1] was developed that described a way of extending BASIC to control and communicate with CAMAC modules.

The existing BASIC interpreter for the Honeywell 516 computer used for experimental data collection was modified to implement this standard. The BASIC interpreter as modified was originally intended to be used to test concepts and methods of using CAMAC modules to control and monitor the accelerator in preparation for the use of a microprocessor-based system with a BASIC interpreter of its own to control the accelerator.

The fortunate acquisition, at a later date, of a Honeywell 316 computer, which executes the same set of machine instructions as the 516 but at three-fifths of the speed, considerably enhanced the usefulness of the modified BASIC interpreter, as the 316 was immediately dedicated to the accelerator control application.

Additional extensions to the BASIC interpreter to

---

[1] *Real-Time BASIC for CAMAC*, ESONE publication RTB/03 or NIM/ERDA publication TID-26619

provide disk file input-output, some limited character manipulation facilities, program chaining and the ability to read the position of the sense switches wre made. These extensions significantly enhanced the potential capabilities of control programs written in BASIC.

CHAPTER V

FORTH

5.1 Introduction

FORTH is an unconventional language for minicomputers that provides a higher-level language capability to systems that may only have a small amount of memory. FORTH, in its present form, was developed in 1971 at the National Radio Astronomy Observatory at Kitt Peak by Charles Moore as the result of experimentation with novel language and implementation concepts that he had engaged in for several years.

FORTH is an extensible language: new syntactic elements of any of the types used in FORTH may be defined by the sophisticated user. All computational commands in FORTH are subroutine calls, thus making the most-used part of the language easily extensible. Assembly-language subroutines can be written within FORTH, and even the list of assembler mnemonics is extensible.

As the extensible nature of FORTH implies that in normal use a significant amount of FORTH code is processed immediately after loading the FORTH system to provide a chosen extended environment to the user, a mass-storage device, such as a disk, is required in any reasonable implementation of FORTH.

FORTH is implemented by means of an interpreter, but it does not provide the same interactive environment as BASIC.

Programs must be listed and edited from the copy in the mass-storage device, not the executable form in the computer's memory.

5.2 Basic Concepts

The basic unit of FORTH is the word. In FORTH, a word is any string of characters bounded by the current delimiter, which is almost always a space. All words must be defined before use, either within the system or by the programmer. Numbers fit this definition of words; they need not be defined explicitly, nor does each number used have an individual definition stored in the computer's memory. A word that is otherwise undefined is accepted as a number if it has the appropriate form, and the action of the word is then that of placing its value on the stack.

Each word is, in effect, a subroutine call. As this means that each word performs a self-contained action, rather than being part of a statement with one of several types of syntax, the language is easily extended by defining new routines. This characteristic of words affects the manner in which arithmetic expressions are entered, requiring the use of Reverse Polish Notation (RPN) and the provision of a stack.

This stack grows from high core towards low core, so that the table of internal word definitions can grow in the other direction, permitting efficient use of all available memory.

The dictionary is the area of memory where the definitions of words are stored in internal form. Each definition has a header containing the partial spelling of the word being defined, in condensed form, a pointer to the header of the previous word in the dictionary, so that the dictionary can be searched, and a pointer to machine-language code somewhere in the computer's memory, which, when executed, performs the action for which the word is a request. Successive headers in the dictionary normally have a number of memory locations between them, where the data that the words are considered to name, or the text of the programs they name, may be stored.

5.3 Language Description

As we have seen, each word in FORTH is a request for action in and of itself. Therefore, arithmetic in FORTH has to be expressed in RPN. As an example of what arithmetic looks like in FORTH, the following FORTH code

```
5 1 + 3 * 2 4 * - .
```

performs the same action as the BASIC statement

```
PRINT (5+1)*3-2*4
```

as the FORTH word . prints the value of the number at the top of the stack and removes that number from the stack.

Some words read the word that follows them rather than allowing that word to be executed. This is done, for example, by words that define new words: the word to be defined cannot yet be executed, as it is not yet defined,

and follows the word which creates its definition.

The most important way of defining a new word is to define it as a call to a FORTH-language subroutine. A typical definition of this type, called a colon definition, looks like this:

```
: XXX DUP 3 - 0< IF 5 * ELSE 4 SWAP - THEN . ;
```

This FORTH code defines XXX as the name of a subroutine; a new FORTH word. This word takes the top number on the stack, and prints five times that number if it is less than three, and prints four minus that number otherwise.

In addition to reading the next word, the colon places FORTH into a special mode, called function definition mode, where words are read and placed into the definition of the new function instead of being executed. Some words, most evidently the semicolon, have to be executed during function definition mode. A bit inside the internal form of the definition of a word is used to indicate such words. The programmer may create new words of this type by defining them in the same way as in a colon definition, but with the word EXECUTE replacing the colon. The words IF , THEN , and ELSE were defined using the word EXECUTE as well as other, more complicated, features of FORTH.

Function definition mode is not required for defining variables, nor is it required for defining assembly-language procedures.

The definition

```
0 CONSTANT XFLAG
```

causes XFLAG to have the action of putting a zero at the top of the stack whenever it is executed. The address of the memory location containing the value of XFLAG can be placed on the stack by the following FORTH code:

```
' XFLAG
```

The word ' reads the word XFLAG, not allowing it to be executed, and finds the memory address in the dictionary where its value is stored, placing that address on the stack.

On the other hand, the definition

```
35 INTEGER W
```

reserves a memory location, initially containing the number 35, and causes the word W to have the action of putting the address of that location at the top of the stack when executed. The contents of that location can be placed on the stack by the following FORTH code:

```
W @
```

After the word W executes, the word @ simply takes the number at the top of the stack, and replaces it with the contents of the memory location of which that number is the address.

The word = is used to assign new values to variables. It takes the number at the top of the stack, and uses it to identify the memory location in which it stores the number second from the top of the stack, and then removes both numbers from the stack. Thus, using XFLAG and W as defined above,

```
XFLAG W =
```

takes the value of XFLAG and assigns it to W, and

```
W @ ' XFLAG =
```

takes the value of W and assigns it to XFLAG. As can be seen, the distinction between INTEGER and CONSTANT type variables is very important.

The word , takes the number at the top of the stack and places it in the dictionary, incrementing the dictionary pointer so that anything new placed in the dictionary does not overwrite the number placed there. One way that this word can be used is:

```
5 INTEGER ARRAY 6 , 7 , 8 , 9 ,
```

Because the word INTEGER always uses the location immediately after the function header of the word it defines for the number to which that word will refer, the code above has the effect of defining the word ARRAY to be a word which places on the stack the address of the first element of an array containing initially the numbers 5, 6, 7, 8, and 9 in that order.

The word CODE begins the definition of an assembly-language procedure. The word CODE reads the word following it, and creates the appropriate header for an assembly-language procedure. This header indicates that the machine code that must be executed to perform the action associated with the new word is located in core immediately after the header.

Creating the header is all that the word CODE does. The

required assembly code must be explicitly placed in the dictionary by the programmer. Before any other words may be defined, it must be ensured, by assembling a jump instruction, that the header of the word following will not be executed as if it were intended as part of the machine-language code in this word.

The process of placing machine instructions in the dictionary is eased by defining all assembler-language mnemonics to be words which have the action of placing the instructions they represent in the dictionary. For the same reason that FORTH arithmetic is done in Reverse Polish form, the operands of memory-reference instructions must precede the instruction mnemonic; first the address is placed on the stack, then the assembler mnemonic takes the address from the stack, combines it with its own opcode, and places the required machine instruction in the dictionary.

A number of words are defined in the system that place jump instructions in the dictionary by means of which an assembler-language subroutine can return control to its caller. More than one such word is defined so that assembler-language subroutines can have a number of common stack manipulations done for them, such as "remove the top two numbers on the stack, and place the contents of the accumulator on the stack instead".

The disk (or other mass-storage device) is where the written form of function definitions can be permanently stored. To the programmer, the disk appears as a set of over

five thousand blocks of storage, each one containing 1024 8-bit characters, or 512 16-bit numbers. The word LOAD takes the top number from the stack, and reads in the disk block with that number, causing the FORTH code in that block to be executed as if it were being typed in on the teletypewriter keyboard. In fact, a disk block being read and executed in this fashion can even contain LOAD commands itself. The disk block must contain, after the text that is intended to be executed, the word ;S which indicates the end of the text in the block.

FORTH code that can be loaded by the command EDIT LOAD (EDIT is a CONSTANT type variable) allows a user to enter text into FORTH blocks, which are considered to have 16 lines of 64 characters, and replace, delete, or move lines. other facilities exist for transferring source blocks to or from paper tape, and for listing any block or group of consecutive blocks on the teletypewriter.

FORTH words have been defined to permit use of the CAMAC system[1]. In addition to assembler-language programs to perform actual CAMAC input-output, a new type of FORTH word, similar to the CONSTANT type, is defined so that the full description of a CAMAC operation, in compressed form, can be placed on the stack by a single easily-defined word.

---

[1] *Go FORTH With CAMAC*, W. K. Dawson, University of Alberta Nuclear Research Center Internal Report 69, July 1974

5.4 The Dictionary

The dictionary, as previously stated, is the area of memory in which FORTH stores the definitions of all the words that have been defined. Each dictionary entry contains a header record that contains the following:

the name of the word being defined, in the form of its first three characters, plus a byte indicating its length,

a pointer to the header of the previous word in the dictionary,

and a pointer to the code which must be executed to perform the action that the word is intended to perform when executed.

Since only partial information about the spelling of a word is retained internally by FORTH, it is necessary to create new words with care.

Most words also have a "value"; some area of memory adjacent to the function header contains either the program text of the word, if it is a subroutine defined by a colon definition or in assembler language, or the numerical value thought of as belonging to the word, as for INTEGER and CONSTANT type variables. Except for some types of character string, the value of the word is always in memory locations following the header.

There are three fundamental classes of FORTH words with respect to the pointer identifying the code that will perform the action expected of the word.

The most straightforward case is that of assembler-language procedures. Here, the code to be executed is the value of the word, and that is precisely what is pointed to for execution.

The second case is that of colon (and EXECUTE) defined words. Here, the pointer points to part of the FORTH interpreter itself, which then reads the FORTH code in compressed internal form and executes it. In internal form, each FORTH word becomes a 16 bit address pointing to its definition, with the exception of numerical constants. As numerical constants are not pre-defined, two words are required to represent them: the first points to a nameless word that reads, and causes to be skipped in execution, the second word, which contains the value of the number. Also, many EXECUTE-defined words place in the dictionary pointers to the definitions of other words, many of which cannot be used directly because they would not be useful, and in some cases also because they are defined within the FORTH system and have no names.

The third case is that of words defined by means of words like CONSTANT and INTEGER. Here, the pointer points to a simple procedure that does something with the value or the address of the value of the word, such as placing it on the stack, or, in the case of assembler mnemonics, in the dictionary. There are many different types of words in this category, and there is also a facility to create new types of words like this, as was done for CAMAC.

The code stored internally in a colon definition is called "indirect threaded code" because, instead of simple threaded code, where a list of pointers to machine-language subroutines is processed, we have pointers to the word definitions, each of which contains a pointer to the required machine-language code. The advantage of indirect threaded code as used by FORTH is that except for the few words which are defined as assembly-language procedures, what makes the difference between different words of the same type is not the machine code used by the word, but the data that that machine code processes. Therefore, one pointer does the job that would otherwise require two: a pointer to the program, and a pointer to the data.

CHAPTER VI

CONTROL REQUIREMENTS

6.1 Introduction

There are two ways in which the computer can be useful in accelerator control: it can itself perform the task of control, or it can assist the person who is controlling the accelerator. Both of these areas are addressed by the research covered in this thesis, with autonomous control as the ultimate goal, and limited control plus operator assistance when human intervention is required as the intermediate goal.

Also, providing operator assistance features first makes it possible for the same capabilities to be used in observing the behavior of the control program during its development.

6.2 Operator Services

There are many ways in which the computer can assist the accelerator operator. Even the simple task of displaying machine parameters on a CRT can be of assistance if the display makes them more accessible, better organized, or if the display shows only those parameters that are currently of interest to the operator.

Some accelerator parameters of interest, such as the final particle energy selected by the analyzing magnet, can only be derived by calculation: this makes the display

genuinely useful, especially when the alternative is using a set of tables.

This simple task can be extended somewhat, by including schematic diagrams, or even a simple forms ruling, in the display. A handy means of selecting which display is desired, without using the teletypewriter, is also important; essentially, what is desired is a device such as a light pen to select an alternative from a menu on the screen. This same facility would be extended to include the other services the computer would provide.

A written record of accelerator operating conditions is kept to ease setting the machine for a comparable desired beam in the future, and to monitor gradual changes in the performance of the accelerator. The computer, having access to most of the machine parameters, can easily produce a printed sheet on the teletypewriter having most, if not all, of the needed information. Also, records of machine parameters can be kept on the disk, where the computer can retrieve them quickly for the operator after searching on the basis of the characteristics of the desired beam. This data base could also be useful to the computer as a reference to the computer when and if the ambitious task of setting up the accelerator by computer is attempted.

The accelerator console is already provided with a generous complement of warning lights and buzzers to alert the operator to problems. However, the computer can also assist in this sphere, both by identifying the specific

problem on paper or on a CRT display, and by detecting unacceptable conditions that require calculation to detect, such as the beam being bent within the analyzing or switching magnets to strike the beam tubing.

Occupying a position intermediate between operator assistance and machine control is changing the particle energy by a small amount upon request, or assisting the operator to do so. The required magnet settings and the Mobley RF level for a given energy are both derived from equations involving square roots or trignometric functions. When these functions are not provided within the programming system used, as is the case in the version of FORTH used in this work, the program to perform these calculations is itself quite bulky, yet, if large changes, including large cumulative changes, in the energy of the beam are permitted, the program for this will have to be integrated with general machine control.

6.3 Accelerator Control

In normal operation, the accelerator system and data collection apparatus are left running on a 24 hour a day basis. Safety considerations require that at least two persons, one of whom is experienced in machine operation, always be present in the building when the accelerator is running. As can be imagined, this leads to some degree of strain upon existing manpower resources. Even partial machine control of the accelerator will not, however, permit

relaxation of the requirement that two persons must be present, because this requirement is for the purpose of ensuring that emergency services can be notified or first aid administered in the case of accidental injury to an experimenter. Thus, a control system that can stand entirely alone is the requirement.

The problem of accelerator control has varying degrees of complexity even when a fully autonomous control system is demanded. Accepting that the accelerator has first been set up manually, there are two main sources of the variation in possible complexity.

The first source is the number of different ways in which the accelerator is used. Depending on whether a pulsed beam is required, on the top end voltage and consequent particle energy, and on the level of beam current required, controlling the accelerator can be easy or difficult, for both the computer and for the human operator. A program which could provide unattended operation under most of the circumstances under which the machine is in use would still be useful, even if manual control were resorted to on some occasions.

The second source of variation in difficulty is the degree of efficiency in the utilization of the accelerator that is desired. Once the ability to safely shut down the accelerator system is achieved, a program that simply shut down the accelerator at the first sign of trouble would allow unattended operation of a sort if the goal of

efficiency is quite modest.

A pulsed beam is almost always required, thus increasing the number of things to control and increasing the demand on the ion source for beam current. While the terminal voltage is seldom as low as two million volts, it is not over 6.5 million volts most of the time, thus the machine is only occasionally taken to its very limits. Thus, accelerator control will normally not be trivial, but neither will it be insuperably difficult.

Perhaps the greatest difficulty is to insure that in no case, up to and including power failure, will catastrophic and expensive damage to the accelerator take place that could have been prevented were there a human present. Currently, the computer's own disk drive is perhaps the system component most vulnerable to the consequences of a power failure.

As far as normal operation is concerned, several possible areas of closed loop control exist which the computer must handle to ensure smooth and effective operation of the accelerator system.

One of the most frequently adjusted controls is known as the delay tune on the Mobley system. This is the control that determines the relationship between the phase of the signal applied to the Mobley RF deflection plates and the signal received from the 10ns capacitative pickoff that precedes these plates in the beam line. Only if this control is properly adjusted can the beam pulses be compressed in

time properly within the Mobley magnet. Most of the other controls associated with the Mobley RF electronics simply manage the business of making a radiofrequency oscillator work; the other main exception, the RF level, has a value which can be determined for any given beam by means of an equation[1].

A simpler, but far less important, area of control is simply keeping the field in the analyzing magnet from slowly drifting by adjusting the analyzing magnet current every now and then. Given enough time, such drift can reach into the third significant figure of the particle energy, which is not usually serious, but keeping an eye on this helps to guard the validity of experimental data.

When the accelerator is being operated under more trying circumstances, the ion source begins to cause problems. The source magnet current is often adjusted in such a case: but, when it is adjusted, the source oscillator current must be carefully monitored so that it does not take values high enough to risk damaging the ion source or the source oscillator. Each ion source has its own adjustable trimmer capacitor which can be adjusted to reduce the need for a high source magnet setting for a given beam intensity. These are controls that will be among the last to be operated by the computer, even after hardware to monitor the source oscillator current for the computer is installed.

---

[1] *A Study of the Structure of the Nuclei P^{29} and P^{31}*, Walter Garfield Davies, PhD Thesis, The University of Alberta, January 1966

Another fairly common control measure taken by operators when some kinds of beam are being used is to manually change the mode of the slit control system. This is because at fairly high terminal voltages, apparent electrical changes at the top end are often perceived by the generating voltmeter, thus causing an unnecessary change within conditional mode from slit control to generating voltmeter control. This is because the slit feedback system switches from slit control to generating voltmeter control when in conditional mode if there is a voltage error of over 30kV, even if there is a beam going through the slits. Being able to manually disable this feature when it is not needed for protecting against a spurious beam, while retaining the other indications for moving to generating voltmeter control when in conditional mode, would solve this problem of unnecessary reversion to generating voltmeter control without recourse to the computer.

CHAPTER VII

HARDWARE CONFIGURATION

7.1 Introduction

This chapter describes the electronic system used for accelerator control, including both the computer system and the interfaces between it and the accelerator. Figure 2 is a schematic diagram of this system as a whole.

7.2 The Computer System

The computer system assigned to the control of the Van de Graaff accelerator and associated beam handling equipment is a Honeywell 316 with 16,384 16-bit words of core memory and a 7 megabyte disk drive with removable media. It has as peripherals a high speed paper tape punch and reader, and a 120 cps teletypewriter terminal.

7.3 The CAMAC System

All links between the computer and the accelerator system are through CAMAC modules installed in a CAMAC system controlled by the computer. A two-crate CAMAC system, based on the Elliott Executive Suite, and expandable to 29 crates is the system used. In the Elliott Executive Suite, one crate, the system crate, contains up to four branch controllers, each one controlling up to seven crates, each of which must contain a standard CAMAC Type A controller or

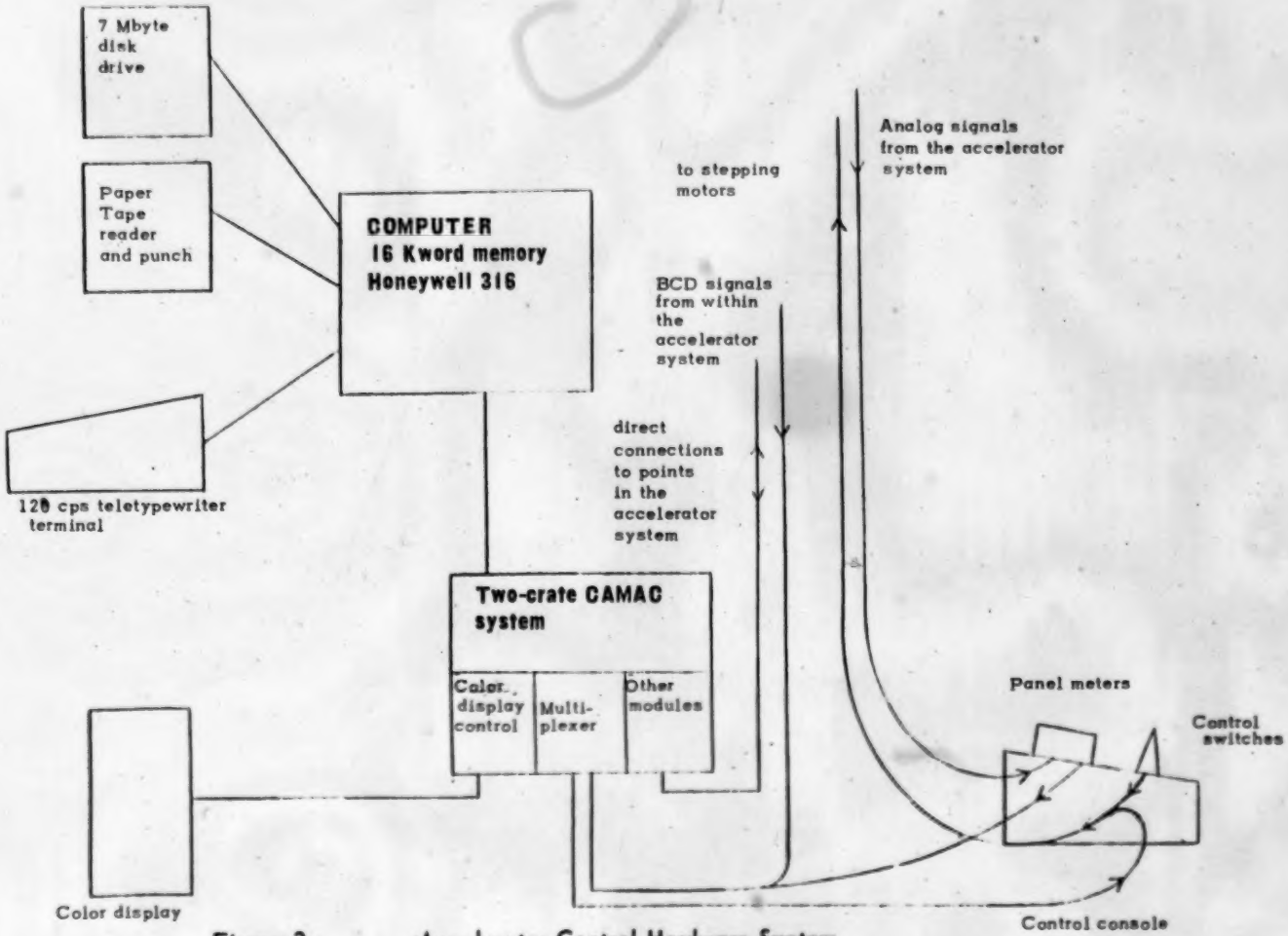

Figure 2: Accelerator Control Hardware System

approximate equivalent thereto, through a standard CAMAC Parallel Highway. The system crate is addressable as though it belonged to a fifth branch, and so ordinary CAMAC modules may also make use of the extra space there.

7.4 Console Electronics

The section of the console most easily accessible to the operator contains a large number of double pole momentary switches, each with an associated panel meter. It is from these switches that most of the machine parameters that must be adjusted remotely are controlled. Each switch controls a stepping motor which, in effect, turns a knob somewhere within the accelerator system. All controls in the top end are controlled through this system, with the stepping motors located below the base of the insulating column of the Van de Graaff, connected by pulleys around which an insulating cord is wound, to the devices in the top end they actually control. Often, as in the case of the top end, it is not practical to make an electrical connection to the devices possessing the parameter controlled. In such a case, the position of the shaft of the stepping motor used to control that parameter is monitored with a potentiometer so that in such cases an indication of the value of the parameter the operator is controlling can exist.

There are many other meters in other parts of the console: several of those are connected directly to CAMAC

A/D converters, whereas the panel meters in the central part of the console are, as they have a digital output, connected to the computer through a digital muliplexer controlled by an especially-designed CAMAC module.

At several points in the beam pipe, the shape of the particle beam can be observed electronically through the beam profile monitor system. Another oscilloscope in the console displays the signal from one of the capacitative pickoffs associated with the Mobley system. As the pulse being displayed is on the order of either one or ten nanoseconds long, a sampling oscilloscope is used to display the signal in order that a slowed-down version of the same signal is available to be digitized for the computer.

7.5 CAMAC Modules

The first module to be examined, together with electronics in the control console, forms a system that, on the one hand, muliplexes the digital outputs of the digital panel meters (and a few other devices with digital outputs) in the console, converting them from BCD (binary-coded-decimal) to true binary and making them available to the computer, and on the other hand controls the stepping motors in the system upon command from the computer. The computer first sends the multiplexer address of the device it seeks to read or control. Once this address, a number from 0 to 63, is sent to the module, subsequent CAMAC commands can read the device with that

address, set the direction in which the stepping motor associated with that device shall move, start it moving or stop it from moving, or even send the device a 24-bit data word. The latter capability, called set point control, is only used with the Mobley magnet current at present.

For computer control of the stepping motors or any other part of the accelerator system to take place, it must be enabled by a switch which can only be operated manually.

A LeCroy 2256S Waveform Digitizer is perhaps the most elaborate item of data collection equipment in the computer control system. It is capable of rapidly sampling up to 1024 data points in an analog signal, converting them with 8-bit resolution, and storing their digital values in an internal memory which can then be interrogated by the computer at its leisure.

This waveform digitizer is used to provide the signals from either the beam profile monitor or the capacitative pickoffs to the computer. In the case of the capacitative pickoffs, the signal received by the waveform analyzer is a slow version of the signal from the pickoffs produced by a sampling oscilloscope. A CAMAC module made in the lab, located beside the waveform analyzer, selects between these two alternatives and provides appropriate level translations and an appropriate clocking frequency to the waveform analyzer. This module also sends a signal to the waveform analyzer to stop continuous sampling and hold the data it has acquired. The module sends this signal after having

received a CAMAC command to do so from the computer: but the signal is not sent immediately. Rather, it is sent upon receipt of a synchronizing pulse from the beam profile monitor or the sampling oscilloscope, thus providing accurate framing of the signal stored in the waveform analyzer for the computer.

The most elaborate repertoire of CAMAC commands in the system is that of the Kinetic Systems module controlling the color CRT display. While this display is fundamentally a text display, up to 256 characters can be defined dot-by-dot by the user and adjacent character zones meet without gaps both horizontally and vertically, therefore it can be used to display graphics.

Six colors, plus black and white, are available. Within the space assigned to any single character, only two colors, called the foreground color and the background color may be present; as the name implies, characters defined in ROM within the display controller are in the foreground color upon a field of the background color. These two colors may be different for each character, as they are specified in the same 16-bit word transmitted to the display controller that selects a character.

Each line of characters across the screen also has characteristics that can be individually specified. While the display controller's internal memory allows 32 lines of 64 characters each, the characters can be widened so that only the first 48, 32, or 16 characters of a line are

visible on the screen. The entire screen comprises 243 horizontal lines for dots; from one to sixteen such lines may be assigned to any line of characters.

In the display controller, characters are defined as matrices that are 16 dots high and 8 dots wide. It is also possible to cause characters to be expanded vertically, so that the eight rows of dots in the bottom half of a character are mapped to two adjacent rows of dots each in the displayed form of the character. This, too, is specified for a line at a time. The line register file, which contains the characteristics of each line, can be disabled, in which case all lines will have the same characteristics, those specified by the first register in the line register file.

The cursor, the position where the next character to be written on the screen will appear unless specified otherwise, can be displayed on the screen as a flashing white square if so commanded by the computer. External signal inputs to the CRT display controller exist by means of which the cursor position may be shifted in each of the four cardinal directions. At first, a box with switches, and now a joystick, have been attached to these inputs so that the operator can communicate with the computer by moving the cursor. The display controller also has an external interrupt input, to which a large red pushbutton has been attached, for use in issuing service requests to the computer.

CHAPTER VIII

OPERATOR INTERFACE

8.1 Introduction

This chapter is concerned with the way the various input and output facilities available to the computer are used in communicating with the accelerator operator.

8.2 Computer Outputs

The most noticeable part of the person-computer interface is the color CRT display, which is normally used to show the names and values of a selection of machine parameters with a schematic diagram of the part of the accelerator system to which the refer.

For this display, a character height of eight rows of dots is chosen, thus making the characters square and thereby enhancing graphics flexibility. Normally, a 48-character line is used, so that these square spaces appear square, except when the amount of information to display requires the full 64-character line.

This diagram, which is usually a schematic of some or all of the accelerator system, but which is in one case merely a forms ruling, has the following two characteristics:

1) Only a small minority of the total character positions on the screen are filled with a nonblank character belonging to the diagram, and

2) The nonblank characters of the diagram tend to be clustered in connected vertical and horizontal lines.

Thus, it would be very inefficient to represent a diagram as a string of characters long enough to fill the screen, most of which are blanks.

Representing a diagram as a set of ordered 3-tuples of the form (X co-ordinate, Y co-ordinate, character) is more reasonable, as it takes the first of the diagram characteristics mentioned above into account, but is still quite inefficient. This approach was used with an early version of the console scanner. A BASIC program was written to generate diagram description files of this form interactively. This program accepted such 3-tuples from the teletypewriter, and then displayed the characters as specified on the screen while writing the 3-tuples in a file. This limited facility eased the detection and correction of errors when preparing such files.

The current BASIC and FORTH versions of the console scanner use a different type of picture description. While the form of the picture description is not the same for the two console scanners, both operate on the same principles.

This type of picture description is more complicated than those mentioned above, but is more compact as it takes both of the diagram characteristics noted above into account. It operates on the principle of representing the diagram as a series of commands. These commands may change

the color used to draw the diagram, or indicate a cursor position from which to draw characters as well as the direction in which to draw future characters from there.

Commands similar to an IF statement make it possible to have the form of some parts of the diagram conditional upon status bits which indicate, for example, which beam line is in use or which beam stops are open.

The picture descriptions defined and used in this manner with the FORTH scanner at present all fit within a single 512-word disk block each, with space left over. Thus, as a buffer the size of a single block is required for data transfers to or from the disk, increased data compaction probably will not motivate any future changes to the picture representation.

A BASIC-language program is used to assist in the preparation of picture descriptions for the FORTH console scanner. The picture description is accepted by the FORTH console scanner in the form of 16-bit words with the first five bits indicating the type of display command, and the remaining eleven bits containing any data that is applicable. The BASIC program produces the picture description in this form from a form resembling assembly language by using mnemonics for the instructions and data in the form of typed numbers.

Another display that the color CRT is used to present is a graph of either the beam profile from the beam profile monitor or the time profile of the pulsed beam from the

capacitative pickoffs, as received from the waveform digitizer. This display is primarily intended for use in software testing, as the same information is already well displayed by oscilloscopes in the console.

For this display, the characters are again chosen to be eight dots wide, but they are chosen to be ten rows of dots high. The characters are divided into two parts laterally, each half either blank or filled, starting from the base, with from one to ten rows of dots to be lighted on the screen. This requires the use of 121 characters in the CRT controller's Symbol Memory, as contrasted with the 64 characters used for the accelerator schematic diagrams.

The other computer output facility is the teletypewriter, used for printing output in the conventional manner. It is used to make a printed report of beam conditions, and the program to do this takes advantage of the tabulator feature within the teletypewriter. This program is driven by a table listing the items to be listed in the main body of the report; it does so by providing the name of the device in its long form for printing as well as the units, if any, of the measurement made, and a single number which points to the entry associated with that device in the device description table, used by the console scanner to access every device in the system. The logout program works from machine parameters as they are stored in core rather than re-reading these parameters directly from the devices. Thus, it can also be used to print out old machine

conditions that have been saved on the disk.

8.3 Computer Inputs

The keyboard of the teletypewriter, used in the conventional manner, plays an important role in allowing the operator to communicate with the computer. Typing input for the computer, even when only the numerical keypad to the right of the full keyboard is used, requires a large amount of attention from the operator. Thus, less conventional means of input are used in the control system.

The external interrupt and external cursor control features of the CRT display controller are used for issuing commands to the console scanner. A button connected to the CRT display controller's external interrupt input allows the user to request service during program execution. As the start button on the Honeywell 316 is used within BASIC and cannot be moved to any position convenient for the operator, it is not used for this purpose.

When service is requested, the computer may display a menu of possible commands on the screen. A joystick or switches connected to the external cursor control inputs of the display controller enable the user to position the cursor to indicate which command is selected.

The four sense switches on the Honeywell 316 console are accessible to user programs from both BASIC and FORTH. Therefore, they are used to shorten some common service requests by allowing the user to set a sense switch

indicating the desired service before issuing a service request to the computer. This enables the step of selecting the service required from a menu on the screen to be bypassed, and by doing so it can also reduce the need for the computer's disk drive to be available.

CHAPTER IX

PROGRAM ORGANIZATION

9.1 Introduction

The BASIC language was used for the first version of the console scanner, a program that senses the machine parameters available to the computer and displays their values on the color CRT display. BASIC programs were also used to test the hardware components of the system as they were added, and concepts used in providing user services and accelerator control were also tested in this way.

As FORTH provides performance greatly increased over that of BASIC, but with decreased ease of use, FORTH is used for the current console scanner. This current console scanner is the repository for the concepts tested previously in BASIC, and this version of the console scanner has an inherent potential for use in accelerator control. Converting the console scanner to FORTH from BASIC resulted in approximately a tenfold increase in its speed, from 8 to 12 seconds to .85 seconds to scan the readings of over 40 devices when control is not attempted. It is the current console scanner written in FORTH that will be described in this chapter.

9.2 The Console Scanner

At its simplest level, the console scanner is a program that reads and displays the values read from a set of devices described in a list. In order that each device can have a fixed reference number within the computer, the device description table contains the descriptions of the devices in a fixed order. At the same time, the order of scanning must be capable of being changed, thus in scanning the various devices the console scanner uses a list of pointers to the entries in the device description table.

The devices must have fixed reference numbers so that control routines can refer to more than one device, and the order of scanning is subject to change for two reasons: so that on the first scan, each device routine has any initial conditions depending on the values read from other devices set for it, and so that the probability of spending time on a low-priority control procedure before discovering the need for a higher-priority one is minimized.

Each entry in the device description table has the information needed to read from a device, to display the value read from it, and to directly control it. Also present is the information on what to do, either in terms of notifying the user or performing control, if the value read from that device fails to satisfy certain conditions, also described in the device description table entry.

Most of this information is in the form of 16-bit binary integers, the form most effectively usable in FORTH.

The information on what method to use for reading a value from the device, what action to take if the value is out of range, how to display the value, and so on, exists only as a number indicating which of a set of subroutines performing that category of action is chosen. Other information, such as device addresses, scaling factors, display color, screen position for display, and limits to acceptable readings from the device are all easily represented as 16-bit binary numbers.

Before scanning, a diagram is drawn on the screen and the names of the devices being scanned are written there; after each scan, a test is made for a user service request in order to service one if present. These additional capabilities of the console scanner were dealt with in the previous chapter.

9.3 Accelerator Control

The concept which lies at the heart of the organization of the control program is a very simple one: any action the computer takes is the result of data received by the computer. Thus, a link must exist connecting the device currently being read to the program that will perform control if that reading indicates a need for control, and a link must exist connecting that program to the devices to be manipulated to perform that control. Until control is actually taking place, links in the other direction are not needed.

The control program must be ready to respond very quickly to problems in the accelerator as they arise. However, it takes time to adjust the controls on the accelerator, and often much longer to see clearly the results of such an adjustment. Since the accelerator is already provided with various simple safety interlock mechanisms that permit it to be safely operated under human control, a net reaction time of about 300ms is adequate, where this time must be sufficient to detect and react to any of the more critical conditions that might take place, but some control activities can take several seconds, or even minutes in some cases. Therefore, it is necessary to have different processes taking place concurrently within the computer.

Concurrency does not, of itself, require the use of interrupts. While interrupts would, in any case, increase the speed of response, they are only absolutely necessary when it is permitted for some programs in the system to be written without taking concurrency into account.

While FORTH posesses an interrupt mechanism, this mechanism only allows interrupts from quiescence, not during program execution. The latter capability, which is what would be required, is difficult to obtain due to the nature of the implementation of FORTH.

Thus, as the required speed does not appear to make mandatory the use of interrupts, their use has not been attempted.

The primary problem that must be dealt with in implementing concurrency of tasks in the control procedure under discussion is the sharing of resources. Normally, the term 'resources' used in this sense refers to tape drives, line printers, disk drives, and so on. Here, however, it is the devices in the accelerator system which are susceptible to adjustment by the computer which are the resources whose allocation is the problem.

The difficulty is due to the fact that these devices are not what are called 'pre-emptible' resources. This means that they cannot be harmlessly borrowed from a task, not currently executing but still active, that is to say, a task that will resume execution again in the future, that is using them. If two tasks seek to adjust the same device at the same time, each task will confuse the other, preventing it from functioning properly.

Thus, a scheme has been developed to permit concurrency while preventing such problems. To see how it works, we again return to the console scanner.

Concurrency is allowed to permit fast response to new problems in the machine even when control is taking place. New problems are detected by scanning the various readings available from the accelerator system, thus, before allowing control tasks to be concurrent with each other, it is necessary to allow scanning to be concurrent with control.

Therefore, before testing to initiate a control procedure in response to the value read from a device, the

console scanner must test to see if that same control procedure, or another one which is the result of having previously read that same parameter, is already in progress. If so, the control procedure already in progress is resumed at this point: when the device that by having an undesirable reading caused the control procedure to begin again comes to be scanned.

Such a scheme requires a complete scan cycle to elapse before a suspended control task can again take over the computer. This is usually sufficient, as the control task itself decides when to allow scanning to resume. But it is not sufficient when a control in the system is actually being adjusted. Yet, such an adjustment takes enough time that scanning must somehow be allowed to continue even then. The special measures needed to deal with this case will be dealt with below.

If a control task was in progress, it is resumed from where it left off. If not, the console scanner can then proceed to test the reading obtained from the device this time. If no need for control is indicated by its value, scanning simply continues.

If a need for control is indicated, the appropriate control routine is entered. This routine must begin by checking to see if any of the devices it will have to adjust are in use by a more important routine. If so, the routine will have to relinquish control to the scanner, and try again next time if the condition persists.

If all the required devices are available, the control routine must now indicate that it is using them. Also, it must set the memory location, called the activity flag, associated with the device that had just been read, so that after the control routine permits scanning to resume the console scanner will, when this device is scanned again, continue with the control routine. Then control can commence.

Three arrays of memory locations are used to allow the protocol described above to take place. Each group of three locations, one from each array, is associated with a single device in the order used for entries in the device description block. As this order is fixed, it is practical for programs to refer to these locations.

The first of these locations has already been named: the activity flag. This location usually contains a zero. When it is not zero, it indicates a control task, initiated because of a reading from the device with which it is associated, is in progress, making it possible for that control task to be resumed.

The second location is called the device pre-emption flag. While the activity flag is associated with the device whose reading started a control procedure, the device pre-emption flag is associated with the devices adjusted by a control procedure. It is zero when a device is not in use. When a device is in use, this flag indicates which routine is using that device by being the number, plus one, of the

activity flag of that control procedure. It may also be a negative number, in which case its absolute value has the same meaning as before, but its sign indicates that the device is not currently in use, but will be used later by the control procedure that has marked it.

The third such location is the priority number. This indicates one aspect of the importance of a control task: how serious it is to stop that task from executing so that another control task can execute. The priority number of a control task has the same position in the array of priority numbers as the activity flag of that task has in its array; thus, only one copy of the priority number need exist. The willingness of a new task to pre-empt other tasks is the complementary facet of the importance of a task, and this quantity is within the program text of the control procedure.

Two other locations, similar to the previous three, also exist. These are required for more involved interactions between tasks.

The first of these is called the mailbox. Through it messages are sent to control activities by other control activities. These locations are associated with the control task that is the intended recipient of the message in the same way as that task's activity flag. In this way, a control procedure that has been pre-empted by another procedure can detect that without examining the device pre-emption flags of the devices it uses. Also, a more

detailed message in the mailbox can, for example, tell such a task how much time it has to terminate in an orderly fashion.

The second location is the future device pre-emption flag. When a sophisticated control procedure has determined that another routine is going to use, but is not actually using, an adjustable device that it wants, makes use of that device without pre-empting the earlier procedure, then a third control routine seeking to use the same device must examine the priorities both of the current user of the device and the control procedure that intends to use it later before attempting pre-emption. Thus, the future device pre-emption flag, usually zero, contains the absolute value of the negative number formerly contained in the device pre-emption flag when a current and future user of the device are both present.

Often, control activities have the form, or some subset of the form:

set A by varying B

and maximize (or minimize) X [,Y,Z...]

by varying P [,Q,R,S,T...]

This general form has several consequences.

B and A sometimes refer to the same device. If they do not, after B is adjusted, possibly by a fairly large step if the required setting can be derived by calculation, there must be a long wait for the machine to settle before reading A.

Normally, B should not belong to the class P,Q,R,S,T. For example, if the computer is told to change the energy subject to maintaining the best possible quality of beam, it is not intended to improve the quality of the beam by causing the accelerator to operate at a lower energy than the one requested. In this connection, however, the computer should reject any requested change that is not feasible: with, perhaps, a password allowing experienced personnel to override the objections of the computer.

When the beam current is to be increased, a more interesting case exists. Until other settings are adjusted for the maximum current on the target, the current resulting from a given source magnet setting is not known. While the source magnet changes the beam current, the goal is to obtain a given current with the lowest possible source magnet setting.

An additional use for the mailbox is suggested by the control activities discussed above. It must be possible to ask a major control activity, such as changing the energy, to slow down until other control activities have dealt with the consequences of its actions, and, in turn, a major control activity may need to be able to request forbearance on the part of other control activities. Eventually, it may be necessary to use more than one word in the computer's memory for the mailbox, but this does not yet appear to be required.

When a device is actually undergoing adjustment, the

computer must be able to stop stepping motor activity within a fraction of a second. For this, resuming a control activity every scan is not fast enough; yet the adjustment takes too much time to stop scanning during it.

Therefore, when a control task is actually performing an adjustment, it will be resumed after each device in the scan is read. No matter how frequently it is chosen to attend to a stepping motor in motion, a problem arises when two are in motion at once.

If it is decided that once a stepping motor comes within a certain time, T, of reaching the desired setting, scanning does not resume, but instead the control routine waits and stops the motor when the desired setting is reached, then such waiting may require a time greater than T; yet, to be sure of stopping the other motor at the right time, the computer must check its position more often than every time T.

Fortunately, there are many ways around the problem. T could be chosen to be small enough that it is not necessary to wait once a motor is within time T of the desired setting, since the motion within time T is tolerable as an inaccuracy in the final setting. Or, once the motor is close to the desired setting, it can then be stopped, and moved to the final setting in a series of short moves which only take place while the program adjusting that motor is executing. Another option is simply to stop all motors but one if more than one motor is within 2T or so of its desired setting.

CHAPTER X

CONCLUSIONS

10.1 Introduction

Software development efforts aimed at computer control of the 7.5 MeV Van deGraaff facility at the University of Alberta Nuclear Research Center have stopped just short of actual accelerator control. In the course of these efforts, the operation of the accelerator, operating practices in its use, and alternative programming techniques were examined.

10.2 Programming Languages

Three language systems are conveniently available at present for use in accelerator control. The first two, BASIC and FORTH, have been discused previously in this thesis. The third is FORTRAN, from which CAMAC is also available by means of CAMAC driver subroutines written to conform to a published standard[1].

Also potentially available for use with the present control computer is the C language devised originally by Bell Laboratories. An effort, reported in a Progress Report from this laboratory[2], to produce a BASIC compiler had been redirected towards the production of a C compiler, which

---

[1] *Subroutines for CAMAC*. Prepared by the U.S. NIM Committee and the ESONE Committee of European Laboratories, U.S. Department of Energy Assistant Secretary of the Environmental Division of Biomedical and Environmental Research publication DOE/EV-0016 UC-37, June 1978.
[2] *Progress Report, 1973*, pp. 92-94, Nuclear Research Center, University of Alberta Department of Physics.

exists in completed form.

BASIC and FORTH have been compared above: BASIC is easier to learn and use, and offers fully the usual conveniences of an interpretive implementation, but is slow in execution, while FORTH is more difficult to use, offers only part of the convenience normally associated with interpreters, but is much faster in execution speed than BASIC.

FORTRAN offers speed comparable with that of FORTH, but lacks entirely the features of interpreters that speed "turnaround" time in debugging. The great advantage of FORTRAN, however, is its familiarity to existing personnel, both programmers and end users. As the subroutines called implicitly by FORTRAN programs for input-output conversion, subroutine parameter passing, and floating-point arithmetic are not written in re-entrant code in this implementation of FORTRAN, it is at least as difficult to implement true interrupts in FORTRAN as in FORTH.

It is expected that programming efforts to control the accelerator by computer will continue over several years, including responses to minor hardware changes and maintenance of the control software. Thus, the advantage of familiarity offered by FORTRAN seems to outweigh the advantages of FORTH for future developments.

10.3 Operational Reliability

Unattended automatic operation of the accelerator system places great demands on the reliablility of the hardware and software system used. Can these demands be met, and what measures are required to meet them?

The reliability of the computer, its peripherals, and the CAMAC interface system has only been fair in experience so far. Some of the failures that have been encountered could have had serious detrimental effects on accelerator control; in this category were, for example, single bit failures on the CAMAC bus. Other failures, primarily those of the computer and the disk, tend to show up when the computer system is powered on, and thus pose less danger. The least reliable component has been the 1200 baud teleprinter, but this is also the least critical component and can be quickly replaced even while the computer is in operation.

At least two problems remain even if perfect hardware reliability can be taken for granted. The first problem is that of producing software that is free of errors and able to cope with any problem that may arise in the accelerator system. The second is a more straightforward problem, that of catastrophic conditions which interfere with computer operation.

The problem of producing error-free code is, of course, one of the perennial problems of the data-processing industry. Various approaches to this problem have been

proposed, many of which contain the concept of building programs out of small, easily understandable parts.

The problem of making the computer able to cope with all possible difficulties with the accelerator has one well-known solution: rather than fruitlessly trying to enumerate all possible problems that may arise, one makes a positive test that indicates that the system is, or is not, operating properly, followed by an attempt, which may safely fail, to find the particular problem. If that attempt does fail, the system is simply shut down until human assistance arrives.

It is not yet clear how such an approach may be adequately implemented. The presence of beam on target, for example, is not satisfactory as a positive test as serious problems can take place that do not disturb the beam, such as excessive source oscillator current, and absence of beam, even if the cause is not identified, does not always indicate a serious problem; many temporary disturbances in the beam are self-correcting and do not require the control program's attention.

The problem of power failure and similar disruptions is less complicated, but is intractable. A potential for expensive damage to the accelerator system is added that did not exist before. Improvements in the accelerator system, such as changes in the vacuum pumps used, make this less serious than before, but the computer has no means of reacting to a power failure. Even a power-fail interrupt

would not be adequate, as the machine cannot be shut down in a few milliseconds. An uninterruptible power source system for the computer and console electronics seems to be, at the least, very desirable.

10.4 Hardware Extensions

In order that the program performing accelerator control can be large enough or involved enough to perform that duty as desired, it may be considered desirable to expand the computer system used for accelerator control.

There are two ways to divide the possible alternatives into two halves: one can use a bigger computer, or more computers; the same kind of computer, or a different kind of computer.

Using a bigger computer of the same kind is straightforward, and presents no unusual problems. However, the alternatives in this direction are limited, as computers with the architecture of the Honeywell 316 minicomputer are no longer being manufactured. The purchase of a used model 516 or 716 computer is possible, or the memory of the existing computer could be expanded. Both approaches suffer from a price/performance deficiency compared to the use of more current technology, and memory expansion of the 16 Kword 316 computer will still require program conversion to some extent, due to the nature of 316 addressing.

The obvious disadvantage of using a larger computer of a different kind is the requirement for program conversion

or redevelopment. The machine-independent nature of CAMAC removes most of the interfacing problems that such a decision would otherwise bring, and many commercial systems of hardware and software for utilizing CAMAC are available. However, as there are no other uses envisaged for the Honeywell 316 computer than that of accelerator control, buying a larger computer would probably be part of a "more computers" alternative, thus allowing continued use of the existing CAMAC drivers, and so on.

When the use of more than one computer for accelerator control is considered, the question of how the task of accelerator control is to be divided among the computers used is raised.

Whether the additional computers are the same as, or different from, the computer currently being used is less important in this case, but there is still an advantage to keeping the number of incompatible architectures in the system to a minimum. In practice, it may well be found that additional computers of both the same and different types will be added.

A Honeywell 316 computer presently in use at the Tri-University Meson Facility (TRIUMF) is scheduled to be returned to the Nuclear Research Center in about a year; so far, accelerator control is the only application that has suggested itself for that computer. This computer has 32 Kwords of memory, and a CAMAC interface of the same type as that of the present 316 computer, but it has only a magnetic

tape drive instead of a disk. Also, it has an extra teletypewriter port, for which software exists to drive a graphics terminal.

As for additional computers of a different kind, this alternative may well be taken almost inadvertently, as more and more CAMAC modules come to include microprocessors to enhance their power and flexibility. Also, using a microprocessor-based intelligent crate or branch controller offers a method of adding computer power to the system that is both convenient and painless, in that no new interfaces need to be sought, either computer-to-computer or computer-to-CAMAC. Clearly, the question of how to partition duties among the computers in the system is profoundly affected by their relative sizes.

What are the different duties in the control system that lend themselves to being partitioned? To answer this question, one can start by preparing a list of the functions in the system that appear to be distinct:

Operating the color display

Generating printed reports of beam conditions

Recording machine conditions on the disk

Conversing with the machine operator

Reading accelerator parameters

Adjusting accelerator controls

Analyzing waveforms from the waveform analyzer, returning a few characteristics of the waveform.

Machine control itself: deciding when, if, and what to

do to the accelerator

Once the different duties in the accelerator control system have been listed in such a way, the next step is to look at the restrictions present when allocating them to different computers.

A fundamental restriction is that adding a new computer to the system and assigning a task to it must decrease the load on the other computers. This will not be true if, for example, a computer spends more time conversing with the computer performing a task than it would doing that task itself, or if the program to converse with the new computer takes up more storage than the program to do the task itself would take. Even in such a situation, an extra computer could be useful if there was an excess of time but a lack of memory space, or the reverse, provided the more plentiful resource is the one wasted.

Other restrictions may stem from the particular hardware and software systems intended to be used. Several restrictions present themselves in the case of the foreseeable extensions to the accelerator control system:

a teletypewriter must be connected to the same computer that is connected to the disk drive, which may or may not be the computer with 32 Kwords of memory,

no more than one teletypewriter may be connected to the system, due to space limitations (an extra teletypewriter is available, and one could be used for loading the system, while another is used for

conversations with the operator),

Any microcomputers in the system will probably be connected for input-output only through CAMAC, and will control some crate space themselves subject to being bypassed by the parent computer, which would be the 316 unless a multi-level hierarcy were implemented.

Subject to these constraints, and recognizing that the division of the control program into parts given above is based upon what seems conceptually separate to humans, and perhaps not what is easily separated within a computer, what can be said about how an extended multi-computer system might be used?

Connecting both computers to CAMAC through their existing interfaces will mean that for each computer there will be at least one crate that only the other computer can access.

At present, the two crates in the system are almost filled with modules. If a third crate were added, one attractive way to partition the system is to have the disk and teletypewriter connected to the computer that also controls the color display controller and perhaps a few other modules in the single crate which it manages. In this way, all conversions to and from human-readable form are performed by one computer.

Unless the high-level decision making for machine control is very complex, as it may well become in time, the

task of analyzing waveforms will produce the largest single computational load in the system. The computer, which would be a larger computer, and thus one of the Honeywell 316s, doing this should be directly interfaced to the waveform analyzer, but not to many other devices. While actions involving the disk and teletypewriter are initiated from the console, the color display is normally being constantly updated. Thus, one is tempted to consider using a microcomputer to deal with the color display, but this is likely to involve a requirement for yet another crate.

Unlike other control duties, maintaining the color display is not critical, and more time per device will be required to determine the need for control and so on once a considerable amount of control is implemented.

One question that has not been dealt with is how the two Honeywell 316 minicomputers are to be connected. Presumably extra teletypewriter interfaces are simple enough, but it is necessary for the computer with the disk to be able to download the other computer. As the computer without the disk has a magnetic tape drive, it can be loaded with a simple monitor program that would allow it to accept programs and control commands through the connection to the other computer. This would not be excessively difficult even with paper tape, if the magnetic tape unit and the disk were connected to the same computer.

Since the two minicomputers are compatible with each other, it is desirable to allow use of as much standard

Honeywell systems software as possible on both computers. This does not prevent the use of two different language systems on the two computers, but it tends to narrow the choice of possible ways to interface them: for example, while BOS, the Honeywell Batch Operating System, will accept object input from an ordinary teletypewriter (expecting it to be paper tape ASR), it requires a disk to operate, and thus cannot be resident in both computers. However, there are other Honeywell operating systems which have the same object format, and thus the non-disk computer may be running under the same system here as it is running under in its present role at TRIUMF in Vancouver.

While it would be premature to predict the exact shape of a future, expanded, system at this time, a possible configuration has been visualized: two computers of the same general type, one with 16 Kwords of memory, taking care of communicating with the operator both in print and on the display, supervising the entire computer system, and using the disk drive both to save data and load the system, and another computer with 32 Kwords of memory, controlling a two-crate CAMAC system and performing most of the duties involved in accelerator control.

Bibliography

Anonymous, *Model 3232 Programmable Color Display Driver Instruction Manual*, Kinetic Systems Corporation, December 1977

Anonymous, *1973 Progress Report*, Nuclear Research Centre, Department of Physics, University of Alberta, Edmonton, pp. 92-94

Anonymous, *Real-Time Basic for CAMAC*, ESONE publication RTB/03 or NIM/ERDA publication TID-26619

Davies, W. G., *A Study of the Structure of the Nuclei P^{29} and P^{31}*, PhD Thesis, The University of Alberta, January 1966

Dawson, W. K., *Go FORTH with CAMAC*, Internal Report 69, Nuclear Research Centre, Department of Physics, University of Alberta, July 1974

Dawson, W. K., *University of Alberta FORTH Users' Guide*, Internal Report 89, Nuclear Research Centre, Department of Physics, University of Alberta

Elliott, J. B., Holm, L., *Machine Procedure Manual*, Nuclear Research Centre, Department of Physics, University of Alberta, December 1969

Institute of Electrical and Electronics Engineers, *CAMAC Instrumentation and Interface Standards*, Institute of Electrical and Electronics Engineers, distributed by Wiley-Interscience, 1976

Livingston, M. S., Blewett, J. P., *Particle Accelerators*, International Series in Pure and Applied Physics, McGraw-Hill, 1962

Newman, W. M., Sproull, R. F., *Principles of Interactive Computer Graphics*, McGraw-Hill Computer Science Series, 1973

Savard, J. J. G., Dawson, W. K., Tate, G. R., *Extensions to BASIC*, Internal Report 97, Nuclear Research Centre, Department of Physics, University of Alberta

United States Nuclear Instruments and Methods Committee, ESONE Committee of European Laboratories, *Subroutines for CAMAC*, Publication DOE/EV-0016 UC-37, U.S. Department of Energy, Asistant Secretary of the Environmental Division of Biomedical and Environmental Research, June 1978

Yourdon, E., *Design of On-Line Computer Systems*, Prentice-Hall, 1972

Appendix I

FORTH Control System Source Code

This appendix contains the source code of the console scanner program as written in FORTH. The disk blocks are numbered both in decimal and in octal. Blocks below 128 (decimal) are not part of the console scanner, but are included for reference purposes as they provide the environment in which the console scanner was written, including the CAMAC commands it uses.

The sequence of FORTH commands

```
6 LOAD 7 LOAD DECIMAL 500 LOAD
```

is used to load the console scanner and the subroutines it calls after FORTH itself has been loaded. To run the console scanner, the commands

```
AINIT SCANNER
```

are required, after which the computer will prompt for the replies to one or two questions, the numerical replies to which are terminated by a carriage return rather than a Control-D.

Only the blocks from block 300 onwards, which have been specifically developed for accelerator control, are documented fully to explain their structure and use. The calling sequences of routines in blocks below block 128 are given in some cases when they are referenced after block 300.

```
  BLOCK NUMBER      5 DECIMAL      5 OCTAL

120 LOAD 121 LOAD 123 LOAD 122 LOAD 113 LOAD 124 LOAD
126 LOAD   ;S
```

```
  BLOCK NUMBER      6 DECIMAL      6 OCTAL

11 LOAD 12 LOAD 13 LOAD 14 LOAD 15 LOAD 16 LOAD 17 LOAD
20 LOAD 21 LOAD 22 LOAD 23 LOAD 24 LOAD 25 LOAD 127 LOAD

0 INTEGER  BLK
: LINE   1  -  17  MIN  40  *  BLK  @  BLOCK    +    ;
: LIST  BLK   =   21   1  DO
    CR   I LINE 2 *  100 TYPE  I  .  LOOP CR ;
: FLUSH BUFFER BUFFER ;

: BREAD BUF @ SWAP BUF = SWAP READ DROP BUF = ;
: BWRITE BUF @ SWAP BUF = SWAP WRITE BUF = ;
: BR4 26000 BRSC ;  : BR0 0 BRSC ;
: SECTOR  1000  HERE 1000 MOD -  SWAP OVER > * DP += ;
36 CONSTANT EDIT
: DISCARD   ;          ;S
```

```
  BLOCK NUMBER      7 DECIMAL      7 OCTAL

120 LOAD 121 LOAD 123 LOAD ;S
```

```
  BLOCK NUMBER      8 DECIMAL     10 OCTAL

11 LOAD 12 LOAD 13 LOAD 14 LOAD 15 LOAD 16 LOAD 17 LOAD
20 LOAD 21 LOAD 22 LOAD 23 LOAD 24 LOAD 25 LOAD 26 LOAD

0 INTEGER  BLK
: LINE   1  -  17  MIN  40  *  BLK  @  BLOCK    +    ;
: LIST  BLK   =   21   1  DO
    CR   I LINE 2 *  100 TYPE  I  .  LOOP CR ;
: FLUSH BUFFER BUFFER ;

: BREAD BUF @ SWAP BUF = SWAP READ DROP BUF = ;
: BWRITE BUF @ SWAP BUF = SWAP WRITE BUF = ;
: BR4 26000 BRSC ;  : BR0 0 BRSC ;
: SECTOR  1000  HERE 1000 MOD -  SWAP OVER > * DP += ;
36 CONSTANT EDIT
: DISCARD   ;          ;S
```

```
  BLOCK NUMBER      9 DECIMAL      11 OCTAL

4000 MR LDA, 10000 MR STA, 26000 MR IMA, 141340 CPU ICA,
14000 MR ADD, 16000 MR SUB, 12000 MR ERA, 6000 MR ANA,
2000 MR JMP, 20000 MR JST, 22000 MR CAS, 24000 MR IRS,
140024 CPU CHS, 140100 CPU SSP, 140500 CPU SSM,
 140040 CPU CRA,  140401 CPU CMA,  140407 CPU TCA,
141206 CPU AOA,   141216 CPU ACA,   140320 CPU CSA,
34000 MR MPY, 36000 MR DIV, 72000 MR LDX, 32000 MR STX,
 201 CPU IAB, 401 CPU ENB, 1001 CPU INH, 11 CPU DXA,
101001 CPU SSC,    101040 CPU SNZ,    101400 CPU SMI,
100001 CPU SRC,    100040 CPU SZE,    100400 CPU SPL,
101100 CPU SLN,    100100 CPU SLZ,    101000 CPU NOP,
SP 0) CONSTANT S) 0 CONSTANT 0  1 CONSTANT 1  13 CPU EXA,
CODE +  S) LDA,  SP IRS,   S) ADD,   PUT
: 1)   40000 +  ;          : E)   40000  +  ;
: S   CONSTANT  ;CODE   4  1) LDA,      S) SUB,   COMMA
: I/O  CONSTANT ;CODE   S) LDA,   4 1) ADD,    COMMA   ;S

  BLOCK NUMBER      10 DECIMAL      12 OCTAL

30000 I/O OCP, 70000 I/O SKS, 130000 I/O INA,
170000 I/O OTA,
41500 S LGL,   40500 S LGR,   41100 S LLL,   40100 S LRL,
41600 S ALS,   40600 S ARS,   41200 S LLS,   40200 S LRS,
41700 S ALR,   40700 S ARR,   41300 S LLR,   40300 S LRR,
CODE -  SP LDX,   1 1) LDA,   0 0) SUB,   BINARY
CODE XOR S) LDA, SP IRS, S) ERA, PUT
CODE AND  S) LDA,   SP IRS,   S) ANA,   PUT
CODE OVER   SP LDX,   1 1) LDA,   PUSH
CODE SWAP   SP LDX,   0 1) LDA,   1 1) IMA,   PUT
CODE ROT SP LDX, 2 1) LDA, 0 1) IMA, 1 1) IMA, 2 1) STA,
NEXT          CODE  DUP S) LDA,   PUSH
CODE @      S) LDX,   0 0) LDA,   PUT
CODE =      S) LDX,   SP IRS,   S) LDA,   0 0) STA,   POP
CODE +=   S) LDX, SP IRS, S) LDA, 0 1) ADD, 0 1) STA, POP
CODE 1+= S) LDX, 0 1) IRS, POP POP
CODE DROP  POP    : HERE DP @ ;       ;S
```

```
  BLOCK NUMBER     11 DECIMAL     13 OCTAL

CODE SS1 CRA, 101020 , AOA, PUSH
CODE SS2 CRA, 101010 , AOA, PUSH
CODE SS3 CRA, 101004 , AOA, PUSH
CODE SS4 CRA, 101002 , AOA, PUSH
: LDB,      LDA,   IAB, ;       : STB,    IAB,   STA, ;
: ADM,      DUP   ADD,   STA,   ;
: -IRS,     DUP   LDA,   ONE SUB,   STA, ;
: IF,       HERE   3000 , ;
: THEN,     HERE   777   AND   SWAP   += ;
: ELSE,     IF,   SWAP   THEN, ;
: BEGIN,    HERE ;
100000 CPU NOT=    2000 MR END,
: ,CODE    CODE HERE 1 - += ;
CODE 2DEEP SP LDX, 0 1) LDA, 2 1) IMA, 1 1) IMA, 0 1) STA,
NEXT     SP 1 + CONSTANT RP  DP 3 + CONSTANT LAST
         OP 2 + CONSTANT DELIM        ;S

  BLOCK NUMBER     12 DECIMAL     14 OCTAL

: INTEGER  CONSTANT ;CODE   0 LDA,  ONE 3 + ADD,   PUSH
: SET    SWAP CONSTANT , ;CODE   4 1) LDA,  5 1) 0) STA,
NEXT    10 BASE SET OCTAL   12 BASE SET DECIMAL
20 BASE SET HEXADECIMAL  EXECUTE OCT OCTAL ;
EXECUTE DEC DECIMAL ;  CODE MINUS  S) LDA,  TCA,  PUT
CODE ABS       S) LDA,   SPL,   TCA,   PUT
CODE */     SP LDX, 1 1) LDA, 2 1) MPY, 0 1) DIV, BINARY.
CODE /MOD      SP LDX,   1 1) LDA,   17 LRS,   0 1) DIV,
               1 1) STB,   IAB,   PUT
: *  1 */ ;      : /   /MOD SWAP DROP ;    : MOD  /MOD DROP ;
CODE MAX   SP LDX,   0 1) LDA,   1 1) CAS,   BINARY POP POP
CODE MIN   SP LDX,   0 1) LDA,   1 1) CAS,   POP POP BINARY
CODE MOVE   SP LDX,   BEGIN,   2 1) 0) LDA,  1 1) 0) STA,
   2 1) IRS,   1 1) IRS,   0 1) IRS, END,   SP IRS,   POP.
;S
```

```
  BLOCK NUMBER      13 DECIMAL      15 OCTAL

1 ,CODE SIGN 0 ,    S) LDA,    4 E) STA,    NEXT
0 INTEGER -DEPOSIT  DEPOSIT  OP LDA,  ONE 1 + SUB, OP STA,
   N IRS,    -DEPOSIT  0) JMP,    0 INTEGER F  55 , 240 ,
60 , 261 , 262 , 63 , 264 , 65 , 66 , 267 , 270 , 71 ,
101 , 102 , 303 , 104 , 305 , 306 ,
CODE CONVERT SP LDA, ONE 1 + SUB, 1 LGL, OP STA, CRA, N STA,
S) LDA, SP IRS, BEGIN, 17 LRS, BASE DIV, N 1 + STA, S) LDA,
BEGIN, IAB, BASE DIV, S) STA, 20 LLL, 0 STA, F 3 + 1) LDA,
-DEPOSIT JST, N 1 + LDA, SZE, SWAP END, S) LDA, SZE, END,
' SIGN LDA, SMI, IF, F 1 + LDA, -DEPOSIT JST, THEN, BEGIN,
N LDA, F SUB, SMI, IF, F 2 + LDA, -DEPOSIT JST, SWAP END,
THEN, OP LDA, AOA, S) STA, N LDA, PUSH
CODE NOSIGN S) LDA, CSA, S) STA, CRA, ACA, ' SIGN STA, NEXT
: . SIGN ABS 0 CONVERT TYPE ;  : .. NOSIGN CONVERT TYPE ;
: ?   @ . ;       ;S

  BLOCK NUMBER      14 DECIMAL      16 OCTAL

: COMPILE   HERE  2 + CPU HERE 1 + , ;
COMPILE ELSE    IC 0) LDA,    IC ADM,    NEXT
COMPILE IF S) LDA, SZE, IF, IC 0) LDA, IC ADM,  POP ,
   THEN,    IC IRS,    POP
COMPILE END S) LDA, SZE, IF, IC IRS, POP THEN, IC 0) LDA,
  IC ADM, POP
EXECUTE IF    IF HERE DUP MINUS , ;
EXECUTE THEN    HERE SWAP += ;
EXECUTE ELSE    ELSE HERE DUP MINUS , SWAP THEN ;
EXECUTE BEGIN HERE ;
EXECUTE END END HERE - , ;

COMPILE DO    RP LDA,    ONE 1 + SUB,    RP STA,    0 STA,
   S) LDA,    0 1) STA,    SP IRS,    S) LDA,    1 1) STA,    POP
;S
```

```
  BLOCK NUMBER      15 DECIMAL      17 OCTAL

EXECUTE DO   DO HERE ;
CODE I   RP 0) LDA,   PUSH
CODE J RP LDX, 2 1) LDA, PUSH
CODE K RP LDX, 4 1) LDA, PUSH
CODE 0= CRA, S) CAS, PUT AOA, PUT CODE 0< S) LDA, 17 LGR,
PUT     : <   - 0< ;        : >   SWAP < ;
COMPILE LOOP   RP LDX,   0 1) LDA,   AOA,   0 1) STA,
   1 1) SUB,   HERE
   SMI,   IF,   IC 0) LDA,   IC ADM,   NEXT
   THEN,   IC IRS,   RP IRS,   RP IRS,   NEXT
COMPILE +LOOP   RP LDX,   S) LDA,   0 1) ADM,
  1 1) SUB, S) ERA, SP IRS, JMP,
EXECUTE LOOP   LOOP  HERE  -   ,  ;
EXECUTE +LOOP    +LOOP  HERE  -   ,  ;
;S


  BLOCK NUMBER      16 DECIMAL      20 OCTAL

: WCH 400 XOR SP @ 1 + COUNT TYPE DROP ; : BL 240 WCH ;
1215 INTEGER CRLF   5000 ,          : CR   CRLF COUNT TYPE ;
: DUMP 7 F = SWAP 77770 AND DO CR CR I . I 10 + I DO I @
DUP 0< .. LOOP 10 +LOOP 0 F = CR ;
: WHERE   LAST @ COUNT 3 MIN TYPE ;
DP 1 + CONSTANT CNTXT
EXECUTE (   251 DELIM =  WORD ;

: VOCABULARY  CNTXT @ 2 - CONSTANT ;CODE 0 LDA, ONE 3 + ADD,
 CNTXT STA, NEXT
: ST DELIM = WORD HERE DUP COUNT 2 / 1 + DP += DROP
CONSTANT ;          : #  243 ST ;
;S
```

```
  BLOCK NUMBER      17 DECIMAL      21 OCTAL

CODE 0CIR S) LDA, BEGIN,  40 OTA, END, POP
CODE 1CIR S) LDA, BEGIN, 540 OTA, END, POP
CODE 2CIR S) LDA, BEGIN, 640 OTA, END, POP
CODE 3CIR S) LDA, BEGIN, 740 OTA, END, POP
CODE ?CIR BEGIN, 1440 INA, END, 10 LRL, SP IMA, ONE SUB,
    SP IMA, S) STA, CRA, 10 LLL, PUSH
CODE BRSC S) LDA, BEGIN, 41 OTA, END, POP
CODE /CNA  CRA, IAB, S) LDA, 4 LRL, SP IRS, S) LDA, 5 LRL,
          SP IRS, S) LDA, 2 LRL, IAB, PUT
HERE 37 , CODE FCNA S) LDA, SP IRS, ANA, S) ADD, PUT
: /NCNA /CNA CONSTANT ;  : CAMOP FCNA CONSTANT ;
CODE ?QV CRA, 1140 SKS, AOA, PUSH
HERE 1 +  CODE ?QS 1140 SKS, NEXT LDA, ABORT
HERE 1 + CODE ?QR 1040 SKS, NEXT LDA, ABORT
;S


  BLOCK NUMBER      18 DECIMAL      22 OCTAL

: /CYC CAMOP ;CODE 4 1) LDA, BEGIN, 240 OTA, END, BEGIN,
1340 SKS, END, NEXT
: /R1 CAMOP ;CODE 4 1) LDA, BEGIN, 240 OTA, END, SP LDA,
ONE SUB, SP STA, BEGIN, 1340 SKS, END, BEGIN, 1640 INA, END,
S) STA, NEXT
: /R1A CAMOP 0 , ;CODE 4 1) LDA, BEGIN, 240 OTA, END, BEGIN,
1340 SKS, END, BEGIN, 1640 INA, END, 5 1) STA, NEXT
: /W1 CAMOP ;CODE S) LDA, BEGIN, 340 OTA, END, 4 1) LDA,
BEGIN, 240 OTA, END, SP IRS, BEGIN, 1340 SKS, END, NEXT
: /W1A CAMOP 0 , ;CODE 5 1) LDA, BEGIN, 340 OTA, END,
4 1) LDA, BEGIN, 240 OTA, END, BEGIN, 1340 SKS, END, NEXT
CODE /IR S) LDA, BEGIN, 240 OTA, END, BEGIN, 1340 SKS,
END, BEGIN, 1640 INA, END, PUT
;S
```

```
  BLOCK NUMBER      19 DECIMAL     23 OCTAL

: /R2 CAMOP ;CODE 4 1) LDA, BEGIN, 240 OTA, END, SP LDA,
ONE 1 + SUB, SP STA, 0 STA, BEGIN, 1340 SKS, END, BEGIN,
1540 INA, END, 0 1) STA, BEGIN, 1640 INA, END, 1 1) STA,
NEXT        : /R2A CAMOP 0 , 0 , ;CODE 4 1) LDA,
BEGIN, 240 OTA, END, BEGIN, 1340 SKS, END, BEGIN, 1540 INA,
END, 5 1) STA, BEGIN, 1640 INA, END, 6 1) STA, NEXT
: /W2 CAMOP ;CODE S) LDA, BEGIN, 140 OTA, END, SP IRS,
S) LDA, BEGIN, 340 OTA, END, 4 1) LDA, BEGIN, 240 OTA, END,
SP IRS, BEGIN, 1340 SKS, END, NEXT
: /W2A CAMOP 0 , 0 , ;CODE 5 1) LDA, BEGIN, 140 OTA, END,
6 1) LDA, BEGIN, 340 OTA, END, 4 1) LDA, BEGIN, 240 OTA,
END, BEGIN, 1340 SKS, END, NEXT
     ;S


  BLOCK NUMBER      24 DECIMAL     30 OCTAL

BASE @ OCTAL
CODE PWORD 102 SKS,  2 OCP,  S) LDA, ICA,
BEGIN,  2 OTA,  END,  ICA,  BEGIN,  2 OTA, END,  POP
CODE POFF  BEGIN,  2 SKS,  END,  102 OCP,  NEXT
: PLDR 24 0 DO 0 PWORD LOOP ;
: PTRL 111777 PWORD PLDR ;
: PBLOCK PLDR 201 PWORD DUP 1001 + SWAP DO I @ PWORD
LOOP PTRL POFF ;      201 INTEGER SOB     111777 ,
0 INTEGER GWORD BEGIN,  1001 INA, END,  ICA,  BEGIN,  1 INA,
END,  GWORD 0) JMP,
CODE RLDR  1 OCP, BEGIN,  BEGIN,  1001 INA,  END,
SOB  ERA,  SZE,  END,  NEXT
HERE 1 + CODE RTRL  GWORD JST,  101 OCP,  SOB 1 + ERA,
SNZ,  NEXT  LDA,  ABORT    CODE RWORD  GWORD JST, PUSH
: RBLOCK RLDR DUP 1001 + SWAP DO RWORD I = LOOP RTRL ;
BASE =  ;S
```

```
  BLOCK NUMBER     25 DECIMAL     31 OCTAL

BASE @ OCTAL
: LBLK 21 1 DO CR I LINE 2 * 100 TYPE I . LOOP CR CR CR ;
# BLOCK NUMBER    # BMSG
: NBLK CR CR BMSG COUNT TYPE BLK @ . CR ;
: LREG DO I BLK = 1 LINE @ 135523 - 0= IF ELSE
NBLK LBLK THEN LOOP ;
: PREG DO I BLK = 1 LINE DUP @ 135523 - 0= IF
DROP ELSE PBLOCK THEN LOOP ;



BASE = ;S

  BLOCK NUMBER     30 DECIMAL     36 OCTAL

DISCARD  REMEMBER  DISCARD       : DISCARD  FLUSH DISCARD ;
BASE @  OCTAL
OP 2 + CONSTANT DELIM
: EMOV  -40 MOVE ;
0 INTEGER TEXT   37 DP +=
: STRING  120240 HERE =  HERE DUP 1 + EMOV
   DELIM =  -1 IN +=  WORD  HERE 1 +  TEXT EMOV ;
: "    42 STRING ;       : (    251 STRING ;
: HOLD  DUP LINE TEXT EMOV ;
: T   CR  HOLD  LINE  2  *  100  TYPE  CR  ;
: R   LINE TEXT SWAP EMOV UPDATE ;
: D    HOLD  20  SWAP  DO  I 1 +
  LINE DUP 40 - EMOV LOOP UPDATE ;
: FF   1 + DUP  17 DO  I LINE  DUP 40 + EMOV -1 +LOOP  R ;
: COPY  SWAP BLOCK 1000 + = UPDATE ;
BASE =          ;S
```

```
  BLOCK NUMBER      72 DECIMAL     110 OCTAL

BASE @ DECIMAL
: WSET DO DUP SM1WR LOOP DROP ;
: ISET DO DUP IMWR LOOP DROP ;
: 0SYM 0 16 0 WSET ;
: 1SYM 255 SM1WR 0 15 0 WSET ;
: 2SYM 0 7 0 WSET 255 SM1WR 0 8 0 WSET ;
: 3SYM 128 16 0 WSET ;
: 4SYM 1 16 0 WSET ;
: 5SYM 128 7 0 WSET 255 SM1WR 0 8 0 WSET ;
: 6SYM 255 SM1WR 128 15 0 WSET ;
: 7SYM 1 7 0 WSET 255 SM1WR 0 8 0 WSET ;
: 8SYM 255 SM1WR 1 15 0 WSET ;
: PCOL FGND @ + BACK @ + ;


BASE = ;S

  BLOCK NUMBER      73 DECIMAL     111 OCTAL

BASE @   OCTAL
: 0LIN 405 PCOL IMWR 402 PCOL 6 0 ISET 407 PCOL IMWR
7 0 DO 402 PCOL 7 0 ISET 407 PCOL IMWR LOOP ;
: 1LIN 403 PCOL IMWR 0 6 0 ISET 404 PCOL IMWR 7 0 DO
0 7 0 ISET 404 PCOL IMWR LOOP ;

: WSYM BR4 0 SAWR 0SYM 1SYM 2SYM 3SYM 4SYM 5SYM 6SYM
7SYM 8SYM ;
: WPICT BR4 0 CRWR 10 0 DO 0LIN 3 0 DO 1LIN LOOP LOOP ;



BASE =  ;S
```

```
  BLOCK NUMBER      74 DECIMAL     112 OCTAL

BASE @ OCTAL
# TVOLTGVERRSLITIBAL ICOR I# 5MSG
: 5PAN 0 LINENO = 0 CHRPOS = 5MSG DLIST ;
CODE 4LR S) LDA, 4 LGL, SMI, HERE 3 + JMP, TCA,
SSM, 4 ARS, PUT
: MSC 0 LINENO = 7 CHRPOS = DO CURPOS 0 11 I /CNA
0 FCNA /IR 4LR  2 / SPNUM LINENO 1+= LOOP ;
: MGO 0 CRWR 5PAN BEGIN 13 6 MSC SS1 END ;



BASE =  ;S
  BLOCK NUMBER      75 DECIMAL     113 OCTAL

BASE @ DECIMAL 6 CONSTANT M    0 INTEGER HOURS
0 INTEGER MINUTES 0 INTEGER DAYS 0 INTEGER TEMP
 0 INTEGER SECONDS -3 INTEGER M3 OCTAL 17 INTEGER C17
12 INTEGER C12
CODE DTOB CRA, TEMP STA, M3 LDX, BEGIN, S) LDA,
4 ALR, S) STA, C17 ANA, TEMP ADD, C12 MPY, IAB, TEMP STA,
 0 IRS, END, S) LDA, 4 ALR, C17 ANA, TEMP ADD, PUT DECIMAL
0 M 0 /CNA 0 /R2 HMS  0 M 1 /NCNA HMS1  HMS1 0 /R1 DREAD
 HMS1 16 /W1 DWRIT
: TOD HMS HOURS = 256 /MOD MINUTES = SECONDS = ;
: DINP DREAD 16 /MOD  DAYS = TEMP = ;
# DAY  HOUR MIN  SEC  1/10 # 5MSG
: 3PAN 16 LINENO = 1 CHRPOS = 5MSG DLIST ;
: FST TEMP @ SECONDS @ MINUTES @ HOURS @ DAYS @ ;
: GDATE TOD DIMP BASE @ FST HEXADECIMAL 5 0 DO CURPOS SPNUM
LINENO 1+= LOOP BASE = ;  : SDATE 3PAN 6 CHRPOS = BEGIN
16 LINENO = GDATE SS1 END ;   BASE =   ;S
```

```
  BLOCK NUMBER      80 DECIMAL     120 OCTAL

BASE @ DECIMAL 23 CONSTANT T    0 T 0 /NCNA 0T
0 T 1 /NCNA 1T   0 T 2 /NCNA 2T   0 T 3 /NCNA 3T
0 T 4 /NCNA 4T   0 T 5 /NCNA 5T   0 T 6 /NCNA 6T
0 T 15 /NCNA 15T
0T 0 /R1 IMRD   1T 0 /R1 CRRD   2T 0 /R1 LRRD
4T 0 /R1 SMRD
0T 16 /W1 IMWR   1T 16 /W1 CRWR   2T 16 /W1 LRWR
3T 16 /W1 LAWR   4T 16 /W1 SM1WR   5T 16 /W1 SM2WR
6T 16 /W1 SAWR
0T 25 /CYC SERS   1T 25 /CYC LERS   2T 25 /CYC CERS
3T 25 /CYC RUP    4T 25 /CYC RDOWN
0T 24 /CYC DEXTL   1T 24 /CYC DERSL   2T 24 /CYC DLFR
3T 24 /CYC DCUR   4T 24 /CYC DEXTCUR
0T 26 /CYC EEXTL   1T 26 /CYC EERSL   2T 26 /CYC ELFR
3T 26 /CYC ECUR   4T 26 /CYC EEXTCUR
BASE =   ;S

  BLOCK NUMBER      81 DECIMAL     121 OCTAL

BASE @ DECIMAL
0T 27 /CYC EXSTL   1T 27 /CYC ERSTL   2T 27 /CYC EROC
0T 8 /CYC EXRTL     1T 27 /CYC ERRTL    15T 8 /CYC KSRTL
0T 10 /CYC EXCL   1T 10 /CYC ERCL
3584 INTEGER HIGHBITS   0 INTEGER LINENO 0 INTEGER CHRPOS
3584 INTEGER FGND       0 INTEGER BACK      0 INTEGER FLASH
HERE 63 , CODE CGET FETCH ANA, BACK ADD, FGND ADD,
FLASH ADD,  PUSH
: WMSG COUNT SWAP DROP 0 DO CGET IMWR LOOP ;
: NOUT CONVERT SWAP IP = 0 DO CGET IMWR LOOP ;
: SPNUM SIGN ABS 0 NOUT ;    : DPNUM NOSIGN NOUT ;
 : CURPOS LINENO @ 64 * CHRPOS @ + CRWR ;
: TVLIST BLK = 17 1 DO I LINE 2 * IP =
64 0 DO CGET IMWR LOOP LOOP ; OCTAL
: BACKS 10000 * BACK = ;    : FGNDS 1000 * FGND = ;
: FLASHS 100000 * FLASH = ;     BASE =   ;S
```

```
  BLOCK NUMBER      83 DECIMAL     123 OCTAL

BASE @ DECIMAL 2 CONSTANT M
0 INTEGER BEG 0 INTEGER TERM : BANDE 1 + TERM = BEG = ;
0 M 0 /NCNA IM0 IM0 17 /W1 WC0 IM0 0 /R1 RC0 IM0 0 /R2 RC0D
0 M 1 /NCNA IM1 IM1 17 /W1 WC1 IM1 0 /R1 RC1 IM1 0 /R2 RC1D
0 M 2 /NCNA IM2 IM2 17 /W2 WC2D  IM2 0 /R2 RC2D
0 M 3 /NCNA IM3  IM3 17 /W2 WC3D  IM3 0 /R2  RC3D
IM0 25 /CYC 1UP    IM1 25 /CYC 2UP    IM2 25 /CYC 12DOWN
IM0 26 /CYC 1MOVE  IM0 24 /CYC 1STOP  IM2 26 /CYC 1SET
IM1 26 /CYC 2MOVE  IM1 24 /CYC 2STOP  IM3 26 /CYC 2SET



BASE =    ;S

  BLOCK NUMBER      84 DECIMAL     124 OCTAL

BASE @ OCTAL 0 CONSTANT BLACK 1 CONSTANT DBLUE
2 CONSTANT GREEN 3 CONSTANT LBLUE 4 CONSTANT RED
5 CONSTANT MAGENTA 6 CONSTANT YELLOW 7 CONSTANT WHITE
CODE 4LR S) LDA, 4 LGL, SMI, HERE 3 + JMP, TCA,
SSM, 4 ARS, PUT
: MUXS BR4 0 LINENO = 25 CHRPOS = 13 0 DO CURPOS 0 3 I
/CNA 0 FCNA /IR 4LR 2 / SPNUM LINENO 1+= LOOP ;
: PANBIG FGNDS BACKS 1PAN 2PAN 3PAN ;
4 INTEGER VBACK 1 INTEGER VFORE
: SCANBIG 0 LINENO = 6 CHRPOS = 0 17 BANDE LUPE GDATE
MUXS 51 57 BANDE LUPE ;
: BIGSCAN PANBIG VBACK @ BACKS VFORE @ FGNDS
BEGIN SCANBIG SS1 END ;


BASE = ;S
```

```
  BLOCK NUMBER     85 DECIMAL     125 OCTAL

BASE @ OCTAL
CODE BSWAP S) LDA, ICA, PUT
0 INTEGER DBLOCK  1000 DP +=
: SMREAD BR4 0 SAWR DBLOCK 1000 + DBLOCK DO
SMRD BSWAP SMRD + I = LOOP ;
: SMWRITE BR4 0 SAWR DBLOCK 1000 + DBLOCK DO
I @ BSWAP SM2WR LOOP ;
: SBNUMBER DBLOCK 1000 + = ;



BASE = ;S
  BLOCK NUMBER     86 DECIMAL     126 OCTAL

BASE @ OCTAL
0 INTEGER BUFF 12 DP +=
: GLOOP BUFF 13 + BUFF DO IMRD I = LOOP ;
: WLOOP BUFF 13 + BUFF DO I @ IMWR LOOP ;
0 25 12 /CNA 32 /CYC EB4  0 25 12 /CNA 30 /CYC DB4
: GINT BR4 ECUR EEXTCUR EEXTL /ENB BR0 EB4 GO ;
: ALLDONE BR0 DB4 BR4 /INH DEXTL DEXTCUR DCUR IOF ;
: CLKIN BR4 DEXTCUR DCUR -20 SP += CRRD SCANBIG CRWR
20 SP += ECUR EEXTCUR CLOCK RETINT ;  BUTTON ATTACH ALLDONE
: NINT BR4 EXRTL ?QS CRRD DUP 7760 AND CRWR GLOOP
2520 CRWR WLOOP EXCL CRWR RETINT ;
RTC ATTACH CLKIN  /BDMD ATTACH NINT
: GOTO BR4 WC1 2MOVE BEGIN DUP RC1 - 177774 AND DUP
0< IF 12DOWN ELSE 2UP THEN END 2STOP DROP ;
BASE =    ;S
```

```
  BLOCK NUMBER      87 DECIMAL     127 OCTAL

BASE @ OCTAL 0 INTEGER M 0 , 0 , 0 ,
0 INTEGER RTC 0 INTEGER BUTTON 0 INTEGET /BDMD
-74 INTEGER MINUTE  41 INTEGER ARMW  0 INTEGER TIME
0 INTEGER PILO IP LDA, M STA, IC LDA, M 1 + STA, OP LDA,
M 3 + STA, RP LDA, M 2 + STA, 1240 SKS, HERE 7 + JMP,
20 SKS, HERE 7 + JMP, 404 SKS, HERE 7 + JMP, BUTTON LDX,
3 1) 0) JMP,
/BDMD LDX, 3 1) 0) JMP, RTC LDX, 3 1) 0) JMP, 24 0) JMP,
: ATTACH SNOOP SWAP = ; PILO INTEGER AD0 7200 INTEGER MASK
CODE GO INH, AD0 LDA, 63 STA, ARMW LDA, 20 OTA, MINUTE LDA,
61 STA, MASK LDA, 440 OTA, HERE 1 - JMP, 440 OCP,
20 OCP, 130 LDA, RP 0) STA, NEXT
CODE CLOCK TIME LDA, AOA, TIME STA, IAB, MINUTE LDA,
61 STA, 20 OCP, NEXT
HERE 1 + CODE IOF 220 OCP, LDA, ABORT
CODE RETINT M LDA, IP STA, M 1 + LDA, IC STA, M 2 + LDA,
RP STA, M 3 + LDA, OP STA, 440 OCP, 25 JMP,  BASE = ;S

  BLOCK NUMBER      96 DECIMAL     140 OCTAL

BASE @ DECIMAL 16 CONSTANT M 0 M 0 /NCNA BPS BPS 8 /CYC TBPS
BPS 9 /CYC CBPS  BPS 26 /CYC EBPS  BPS 24 /CYC DBPS
BPS 2 /R1 RBPS BPS 1 /R1 S1BPS BPS 0 /R1 S0BPS
: GSET BR4 0 SAWR 8 0 DO 8 0 DO 16 0 DO K I >
IF 240 ELSE 0 THEN J I > IF 15 + THEN SM1WR LOOP LOOP LOOP ;
: LFR 0 LAWR 27 0 DO 48 LRWR LOOP 32 27 DO 120 LRWR LOOP ;
0 INTEGER BBEAM 1027 DP += BBEAM 2 + CONSTANT BEAM
0 INTEGER CVAL : GVAL CVAL @ - 0 MAX 7 MIN ;
: SAMPLE BR4 CBPS EBPS BEGIN TBPS ?QV 0= END 1024 BEAM +)
BEAM DO RBPS I = LOOP DBPS ;
: XPLO BR4 0 CRWR 182 CVAL = 27 0 DO BEAM 1024 + BEAM DO
0 I 8 + I DO I @ + LOOP 11 / GVAL 8 * 0 I 16 + I 8 + DO
I @ + LOOP 11 / GVAL + 256 + BACK @ + FGND @ + IMWR
16 +LOOP -7 CVAL += LOOP ;

BASE  = ;S
```

```
  BLOCK NUMBER      97 DECIMAL     141 OCTAL

BASE @ DECIMAL    0 INTEGER CNT
: PLOT BR4 CHRPOS = 0 LINENO = CURPOS 182 CVAL = 27 0 DO
DUP DUP 64 + SWAP DO I @ 8 * 11 / GVAL 8 * I 1 + @ 8 * 11 /
GVAL + 256 + FGND @ + BACK @ + IMWR 2 +LOOP LINENO 1+=
CURPOS -7 CVAL += LOOP DROP ;
: PSUM 0 CVAL = 0 CNT = BEAM 1024 + BEAM DO I @ IF
CVAL @ IF 1 + SWAP I @ + SWAP ELSE 1 CVAL = I BEAM -
DUP I @ SWAP CNT 1+= THEN ELSE 0 CVAL = THEN LOOP ;
: LSUM CNT @ 0 DO CR ROT . . . LOOP ;
: APLOT BEAM + 0 PLOT ;
: BPLOT BEAM + 32 PLOT ;
: CPLOT DUP APLOT 64 + BPLOT ;


BASE = ;S

  BLOCK NUMBER      98 DECIMAL     142 OCTAL

BASE @ DECIMAL 15 CONSTANT M
0 M 0 /CNA 25 /CYC HGO  0 M 1 /CNA 25 /CYC VGO
: SAMPLE BEGIN TBPS ?QV 0= END 1024 BEAM + BEAM DO
RBPS I = LOOP DBPS ;
: HSAM BR4 CBPS EBPS HGO SAMPLE ;
: VSAM BR4 CBPS EBPS VGO SAMPLE ;
10000 INTEGER COUNT  : HLUP BEGIN VGO COUNT @ 0 DO LOOP
SS1 END ; : VLUP BEGIN VGO COUNT @ 0 DO LOOP SS1 END ;
: DPLOT RED FGNDS APLOT GREEN FGNDS BPLOT ;
0 INTEGER HPP 0 INTEGER VPP
: HLOOP BEGIN HSAM VPP @ HPP @ DPLOT SS1 END ;
: VLOOP BEGIN VSAM VPP @ HPP @ DPLOT SS1 END ;


BASE = ;S
```

```
***    BLOCK NUMBER    300 DECIMAL    454 OCTAL

  1  529 LOAD 530 LOAD 531 LOAD 532 LOAD
  2  533 LOAD 534 LOAD 535 LOAD
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13  YDRAW
 14  MT1
 15  BASE = ;S
 16  BASE @ DECIMAL  REMEMBER MT2
```

This block, when loaded, loads the blocks containing the program to draw a schematic diagram on the color display, draws such a diagram, and then unloads the code that it has loaded. The word YDRAW , executed in this block and defined in one of the blocks it loaded, performs the drawing.

```
***    BLOCK NUMBER    301 DECIMAL    455 OCTAL

  1  523 LOAD 524 LOAD 525 LOAD 526 LOAD
  2  527 LOAD 528 LOAD
  3  ' HSAM 4 - INTEGER GADAR ' VSAM 4 - , ' FSAM 4 - ,
  4
  5
  6
  7
  8
  9
 10  SWAP
 11  GADAR SWAP SEL
 12  FM2 COUNT TYPE
 13  2 V? VAPLO
 14  MT2
 15  BASE = ;S
 16  @
```

This block, when loaded, loads the blocks which contain the program to take the digitized form of a waveform from the waveform analyzer, and then draw that waveform on the screen, then it does so, and finally unloads the program that it has loaded.

```
***     BLOCK NUMBER    400 DECIMAL    620 OCTAL

  1  BASE @ OCTAL
  2  100000 CPU SKP, 5 CPU SGL, 7 CPU DBL,
  3  : EQ - 0= ;
  4  CODE NOT S) LDA, CMA, PUT
  5  CODE OR S) LDA, SP IRS, S) IMA, CMA, S) IMA,
  6  S) ANA, S) ERA, CMA, PUT
  7  : MVH MINUS MOVE ;
  8
  9
 10
 11
 12
 13
 14
 15  BASE = ;S
 16
```

This block contains a few of the useful words also defined early in the code written for the console scanner.

```
***    BLOCK NUMBER    402 DECIMAL    622 OCTAL

 1  BASE @ DECIMAL  24 LOAD 25 LOAD  OCTAL
 2  #  ** BLOCK NUMBER # B1MS  #  DECIMAL  # B2MS
 3  #  OCTAL# B3MS
 4  : NABLK BASE @ CR CR B1MS COUNT TYPE BLK @ DUP
 5  DECIMAL 4 F = . B2MS COUNT TYPE OCTAL 5 F =
 6  . B3MS COUNT TYPE 0 F = BASE = CR ;
 7  : LISTREGION F @ 2DEEP DO I BLK = 1 LINE @
 8  135523 - 0= IF ELSE NABLK LBLK
 9  4000 0 DO 3 4 5 * * DROP LOOP THEN
10  LOOP F = ;
11
12
13
14
15
16  BASE = ;S
```

This block contains a program to list the source blocks on a given region of the disk, with a format more attractive than that provided by the word LREG in the basic system. One of the important features of this format is that block numbers are given in both octal and decimal. To list the source blocks from whose numbers run from m to n, the command sequence

n+1 m LISTREGION

is used. Blocks whose first two bytes are the characters ;S (in even parity ASCII, forming the octal word 135523) are considered to be empty, and are not printed.

```
***    BLOCK NUMBER    403 DECIMAL    623 OCTAL

 1  BASE @ DECIMAL
 2  0 CONSTANT I1   ' I1 CONSTANT J1
 3  # '# OCM #   # SPM
 4  : DLL 1000 0 DO 3 4 5 * * DROP LOOP ;
 5  : CBLIST CR 7 F = 0 J1 = BASE @ BEGIN
 6  DECIMAL I1 . SPM COUNT TYPE I1 ENQ DARRA
 7  I1 32 * + DUP OCTAL ? DECIMAL DUP 1 + @ DUP 2 - ABS 1
 8  EQ IF OCTAL OCM ELSE SPM THEN COUNT TYPE DUP 2 + ?
 9  DECIMAL DUP 5 + ? DUP 20 + ? DUP 21 + ?
10  OCTAL DUP 26 + ? DUP 31 + ? CR
11  J1 1+= DLL 32 + @ 8 EQ 0= END BASE = ;
12
13
14
15
16  BASE @ = ;S
```

This block defines the word CBLIST , the purpose of which is to print out the current contents of the device description table from memory, thereby assisting in making modifications to this table.

Note that the word ? is equivalent to the two words @ . which print the contents of a memory location given its address.

```
***    BLOCK NUMBER    404 DECIMAL    624 OCTAL

 1  BASE @ OCTAL
 2  100000 CPU SKP, 5 CPU SGL, 7 CPU DBL,
 3  : EQ - 0= ;
 4  CODE NOT S) LDA, CMA, PUT
 5  CODE OR S) LDA, SP IRS, S) IMA, CMA, S) IMA,
 6  S) ANA, S) ERA, CMA, PUT
 7  DECIMAL    : MVH MINUS MOVE ;
 8  0 INTEGER DARRA 2048 DP +=
 9  0 INTEGER LARA  1024 DP +=
10  515 LOAD 450 LOAD 452 LOAD 454 LOAD
11  : SULA 1993 LARA BREAD 1994 LARA 512 + BREAD ;
12  : WTLA 1994 LARA 512 + BWRITE 1993 LARA BWRITE ;
13
14
15  BASE = ;S
16
```

This block contains:

a) Some of the useful words defined at the start of the console scanner code,

b) The definition of an array in memory to be used for storing the device description table, having the same name, DARRA , as is used for the array in the console scanner code for that purpose,

c) Words that load the aforementioned array with the device description table from the disk, and which write the array out to the disk as the new device description table.

Either this block, or the console scanner itself, may be loaded before block 403 and/or block 450 to permit listing and editing the device description table on the disk.

```
***    BLOCK NUMBER    450 DECIMAL    702 OCTAL

 1  BASE @ DECIMAL
 2  : WTUA 4 0 DO 1999 I - DARRA 512 3 I - * +
 3  BWRITE LOOP ;
 4  0 INTEGER DWOR  : CURR 32 * DARRA DWOR @ + + ? ;
 5  : RPL SWAP 32 * DARRA DWOR @ + + = ;
 6  : ENQ 32 * DARRA 22 + + 2 * 6 TYPE ;
 7  : NRE ILIN 32 * DARRA 22 + + 3 0 DO
 8  DUP I + ITEX I 2 * + DUP 1 + @ SWAP @
 9  256 * + SWAP = LOOP DROP ;
10
11
12
13
14
15
16  BASE = ;S
```

This block contains several words used in editing the device description table when in memory.

```
***    BLOCK NUMBER    452 DECIMAL    704 OCTAL

 1  BASE @ DECIMAL
 2  : LENQ 12 * LARA 1 + + 2 * 16 TYPE ;
 3  : UEN ILIN 12 * LARA 9 + + 3 0 DO
 4  DUP I + ITEX I 2 * + DUP 1 + @ SWAP @ 256 * + SWAP =
 5  LOOP DROP ;
 6
 7
 8
 9
10
11
12
13
14
15
16  BASE = ;S
```

This block contains some words used to edit and examine the block in memory containing the long names and units used for printed logouts of accelerator conditions.

```
***    BLOCK NUMBER    500 DECIMAL    764 OCTAL

 1  BASE @ DECIMAL
 2  85 LOAD 24 LOAD 25 LOAD 536 LOAD 501 LOAD
 3  : EQ - 0= ;  : DELY ;  1 F =
 4  512 LOAD  : .X . CR ;
 5
 6  CODE NOT S) LDA, CMA, PUT
 7  : OR NOT SWAP NOT AND NOT ;
 8  513 LOAD 514 LOAD
 9  OCTAL 100000 CPU SKP, DECIMAL
10  515 LOAD 516 LOAD
11  522 LOAD
12
13
14  : MLOAD DO I LOAD LOOP ;
15  599 537 MLOAD
16  BASE = ;S
```

This block loads the FORTH console scanner system in its present form, and defines a few basic words missing from the standard system, such as AND , OR , NOT , and some assembler mnemonics.

```
***    BLOCK NUMBER    501 DECIMAL    765 OCTAL

  1  BASE @ DECIMAL  0 16 0 /NCNA BPS  BPS 8 /CYC TBPS
  2  BPS 9 /CYC CBPS  BPS 26 /CYC EBPS  BPS 24 /CYC DBPS
  3  BPS 2 /R1 RBPS BPS 1 /R1 S1BPS BPS 0 /R1 S0BPS
  4  : GSET BR4 128 SAWR 11 0 DO 11 0 DO 16 0 DO K I >
  5  IF 240 ELSE 0 THEN J I > IF 15 + THEN SM1WR
     LOOP LOOP LOOP ;
  6  : LFR 0 LAWR 27 0 DO 88 LRWR LOOP 32 27 DO 120 LRWR
     LOOP ;
  7  0 INTEGER BBEAM 1027 DP += BBEAM 2 + CONSTANT BEAM
  8  0 INTEGER CVAL  : GVAL CVAL @ - 0 MAX 10 MIN ;
  9  : SAMPLE BEGIN TBPS ?QV 0= END 1024 BEAM +
 10  BEAM DO RBPS I = LOOP DBPS ;
 11  0 15 0 /CNA 25 /CYC HGO  0 15 1 /CNA 25 /CYC VGO
 12  0 15 2 /CNA 25 /CYC FGO
 13  : FSAM BR4 /Z CBPS EBPS FGO SAMPLE ;
 14  : HSAM BR4 CBPS EBPS HGO SAMPLE ;
 15  : VSAM BR4 CBPS EBPS VGO SAMPLE ;
 16  BASE  = ;S
```

This block contains subroutines used in drawing waveforms on the screen. The word GSET causes the characters whose positions in the user-defined symbol storage are from 64 to 191 to be those used for drawing such waveforms, the word SAMPLE reads the 1024 points of a waveform from the waveform analyzer, and the words HGO , VGO , and FGO select horizontal beam profile, vertical beam profile, and pulsed beam shape respectively to be read by the waveform analyzer.

```
***    BLOCK NUMBER    512 DECIMAL   1000 OCTAL

  1  BASE @ OCTAL 5 CPU SGL, 7 CPU DBL,
  2  21 SECTOR , 3 ,CODE ++X
  3  0 , 0 , 0 , S) LDA, IAB, SP IRS, S) LDA, DBL,
  4  ' ++X 1 + 2 / 2 * DUP STA, SGL, SP IRS, S) LDA,
  5  IAB, SP IRS, S) LDA, DBL, ADD, SGL, S) STA, IAB, PUSH
  6  CODE FIND SP LDA, S) ADD, 0 STA, 0 1) LDA, PUT
  7  : ++ SWAP 3 FIND 5 FIND ++X SWAP ROT DROP ROT DROP ;
  8
  9  : PRINT CR COUNT TYPE CR ;
 10
 11
 12
 13  BASE = ;S
 14
 15
 16
```

This block defines a double precision add as the word ++ . The calling sequence of ++ is

a b c d ++

and its result is (d,c)+(b,a), stored with the most significant word at the top of the stack. Also defined is the word FIND , whose calling sequence is

n FIND

which replaces n with a duplicate of nth word from the top of the stack, counting n: for example,

4004 3003 2002 3 FIND

places 3003 at the top of the stack, while leaving 2002, 3003, and 4004 on the stack below it.

```
***    BLOCK NUMBER    513 DECIMAL   1001 OCTAL

  1  BASE @ DECIMAL
  2  : NCV DUP IF 8 + 1024 * THEN ;
  3  : /FCMD 2DEEP 0 2DEEP /CNA ROT 0CIR ROT NCV BRSC
  4  SWAP FCNA ;
  5
  6
  7
  8  0 CONSTANT NOIR
  9  1 CONSTANT DBLUE  2 CONSTANT GREEN  3 CONSTANT LBLUE
 10  4 CONSTANT RED  5 CONSTANT MAGENTA  6 CONSTANT YELLOW
 11  7 CONSTANT WHITE
 12
 13
 14
 15
 16  BASE = ;S
```

This block defines the word NCV , used to convert a branch number to the internal form used by the CAMAC interface, the word /FCMD , which has the calling sequence

branch crate station subaddress /FCMD

and which immediately selects the crate and branch requested, also placing on the stack the station number, subaddress, and the crate index register number (the crate index register is an internal part of the particular CAMAC interface used, and stores a crate number) combined in the form used by several FORTH words which access CAMAC.

```
***    BLOCK NUMBER    514 DECIMAL   1002 OCTAL

 1  BASE @ DECIMAL
 2  : CINIT BR4 1 OCIR ;
 3  BEAM CONSTANT BLARR
 4  : DGCS 1992 BLARR BREAD 0 SAWR 513 0 DO BLARR I +
 5  @ SM2WR LOOP ;
 6  CODE FOD S) LDA, 13 LGR, PUT
 7  0 CONSTANT IX OCTAL 7400 CONSTANT COLR
 8  17777 CONSTANT DMASK 400 CONSTANT CHSE DECIMAL
 9  0 INTEGER IPE CODE SUP S) LDA, 9 LGL, PUT
10  : AWORD ' IX 1+= BLARR IX + @ ;
11  : SEBP BLARR IX + 2 * 1 + IPE = ;
12  CODE ECXC S) LDA, ICA, PUT
13  : FET IPE @ 2 /MOD @ SWAP
14  IF ELSE ECXC ' IX 1+= THEN IPE 1+= 255 AND ;
15
16  BASE = ;S
```

This block defines the word CINIT , which selects crate 1 of branch 4, the word DGCS , which loads the first 64 characters of the color display's Symbol Memory with the characters used in drawing scematic diagrams of the accelerator system, reading these characters from disk block 1992, the word FOD , which returns the first three bits of the word at the top of the stack, the word AWORD , which takes a word of diagram description from the array containing the instructions for drawing the diagram, the word FET , which acts like the word @ , except that it takes a single byte from memory, given a byte address, and several variables used to store information about the appearance of the diagram being drawn, such as the character set (CHSE), or the (foreground) color of characters being drawn (COLR).

Also, this block defines the array BEAM , used to store

a waveform from the waveform analyzer, to occupy the same storage locations as the array BLARR , used for reading FORTH blocks from the disk.

```
***    BLOCK NUMBER    515 DECIMAL   1003 OCTAL

  1  BASE @ DECIMAL
  2  0 INTEGER ITEX 80 DP +=
  3  33 SECTOR
  4  HERE 1 - CONSTANT IXAD
  5  -80 INTEGER MLEN  OCTAL  215 , 337 ,
  6  CODE ILIN BEGIN, MLEN LDX, 104 SKS, HERE 1 - JMP,
  7  4 OCP, BEGIN, SWAP 1004 INA, HERE 1 - JMP,
     MLEN 2 + CAS,
  8   IXAD 1) STA, SKP, END, MLEN 1 + ERA,
     MLEN 1 + LDA,
  9   SNZ, NEXT 0 IRS, END, IXAD STA, NEXT
 10  # LINE DELETED# LDM
 11  215 CONSTANT K215  DECIMAL
 12  IXAD 80 - CONSTANT ITEX
 13  : ILIN BEGIN ILIN CR ITEX @ K215 EQ IF LDM COUNT TYPE
 14  CR 1 ELSE 0 THEN END ;
 15
 16  BASE = ;S
```

This block defines the word ILIN , which accepts a line of characters up to 81 characters long, terminated by a carriage return, from the teletypewriter. The underscore () may be used as a delete line character for correcting errors in input; input of a null line causes the message "LINE DELETED" to be printed.

```
***    BLOCK NUMBER    516 DECIMAL    1004 OCTAL

  1  BASE @ OCTAL
  2  0 INTEGER LCP
  3  : NIP ITEX LCP = ;
  4  : USGRAB 0 BEGIN LCP @ @ DUP 257 > IF DUP 261 BASE @
     12 MIN + <
  5  IF 260 - SWAP BASE @ * + 1 ELSE 0 THEN ELSE 0 THEN
     LCP 1+= END
  6  DUP 240 EQ IF BEGIN DROP LCP 1+= LCP @ @ DUP 240 EQ END
  7  215 EQ IF 0 ELSE LCP -1 += 1 THEN ELSE 215 EQ 0= THEN ;
  8  : NGRAB BEGIN LCP @ @ DUP 240 EQ IF DROP LCP 1+= 1 ELSE
  9  DUP 255 EQ IF DROP LCP 1+= 0 USGRAB 2DEEP - SWAP 0 ELSE
 10  253 EQ IF LCP 1+= THEN USGRAB 0 THEN THEN END ;
 11  # ?:# NMES  : NLIN NMES COUNT TYPE ILIN NIP ;
 12  : ?? NLIN NGRAB DROP ;
 13  : !? NLIN BEGIN NGRAB END ;
 14  0 INTEGER NNS  : V? DUP NNS = NLIN 0 DO NGRAB IF ELSE
 15  I NNS @ 1 - - IF NLIN THEN THEN LOOP ;
 16  BASE = ;S
```

This block defines the words ?? , !!? , and V? , which make use of the word ILIN defined in the previous block to allow the input of numbers.

Each word prompts the user for input with the characters "?:", then accepting a line of input. The word ?? always places only one number on the stack; the word !!? places all the numbers entered on the given input line on the stack.

The word V? has the calling sequence:

n V?

and always places exactly n numbers on the stack, prompting the user again whenever there are still not enough numbers typed in.

```
***     BLOCK NUMBER    522 DECIMAL    1012 OCTAL

  1  BASE @ DECIMAL    CODE ** 0 ,
  2  CODE *D S) LDA, SP IRS, S) MPY, S) STA,
  3  SP IMA, ONE SUB, SP IMA, IAB, PUT
  4  : ++= 2DEEP 3 FIND DUP @ SWAP 1 + @ ++X
  5  ROT DUP 2DEEP 1 + = = ;
  6  CODE DDV S) LDX, SP IRS, 1 1) LDA, IAB, 0 1) LDA,
  7  S) DIV, PUT
  8  : DZ DUP 0 SWAP = 0 SWAP 1 + = ;
  9  : @/D SWAP DDV ;
 10  28 CONSTANT !ST 117 CONSTANT !LE !LE !LE + CONSTANT !RC
 11  CODE SW1 S) LDA, SP IRS, S) ADD, 0 STA, 0 1) LDA,
     IC 0) STA,
 12  POP  31 INTEGER LIM  : SEL 0 MAX LIM @ MIN SW1 ** ;
 13  0 INTEGER DARRA 2048 DP +=
 14  4 INTEGER GCOL  9 INTEGER STATUS  1 INTEGER DIAGRAM
 15  0 INTEGER DCODE  0 INTEGER IJ
 16  BASE = ;S
```

This block defines the word DZ , which takes an address from the stack and sets the location it specifies and the one following to zero, the word ++= , which adds a single precision number to a double precision number on the stack, and stores the result in memory, the word *D , which places the double-precision product of two single-precision numbers on the stack, and the word @/D , which performs divides a double precision number by a single precision number, with the calling sequence:

```
addr spnum @/D
```

where addr is the address of the double precision number to be divided.

Also, this block defines the array DARRA , which is used to store the device description table, and the word SEL .

The word SEL is used to select one of a number of FORTH words,. specified in a special type of list, to be executed.

```
***    BLOCK NUMBER    523 DECIMAL   1013 OCTAL

  1  BASE @ DECIMAL
  2  0 INTEGER TOTL DP 1+= 0 INTEGER ARE
  3  : BANAL 0 ARE = TOTL DZ DO BEAM I +
  4  @ DUP I *D TOTL ++= ARE +=
  5  LOOP ARE @ DUP . TOTL DDV . ;  0 CONSTANT BOT
  6  : ANALY 0 ARE = TOTL DZ DO BEAM I + @
  7  BOT - 0 MAX DUP I *D TOTL ++= ARE += LOOP
  8  ARE @ DUP . TOTL DDV . ;
  9
 10  : BSAV DUP 1 + BEAM 512 + BWRITE BEAM BWRITE ;
 11  : BRES DUP BEAM BREAD 1 + BEAM 512 + BREAD ;
 12
 13
 14
 15
 16  BASE = ;S
```

This block defines BANAL and ANALY , both of which are used to measure the area of segments of waveforms taken from the waveform analyzer; the calling sequence is

lastch+1 firstchannel ANALY

or lastch+1 firstchannel BANAL

each one returns the average height of the waveform from firstchannel to lastch inclusive, but ANALY subtracts a fixed threshold value from the entire profile, this value being contained in the variable BOT .

```
***    BLOCK NUMBER    524 DECIMAL   1014 OCTAL

 1  BASE @ DECIMAL
 2  0 INTEGER COUNTS 255 DP +=
 3  : ZRA 0 DO DUP I + 0 SWAP = LOOP DROP ;
 4  : SBM COUNTS 256 ZRA 1024 0 DO BEAM I + @
 5  0 MAX 255 MIN COUNTS + 1+= LOOP ;    0 INTEGER NPS
 6  : PCF 1024 100 */ NPS = -1 IJ = BEGIN IJ 1+= NPS
 7  @ COUNTS IJ @ + @ - DUP NPS = 0< 0= END IJ @ ;
 8  0 INTEGER THRESH 0 INTEGER QI 0 INTEGER QQI
 9  : EDGE 256 DUP BBEAM = BBEAM 1026 + =
10  0 DUP BBEAM 1 + = BBEAM 1027 + = ;
11
12
13
14
15
16  BASE = ;S
```

This block defines the words SBM and PCF , and the array COUNTS . The word SBM examines the waveform data stored in BEAM , and adds one to the Nth storage location in the array COUNTS for each channel of BEAM whose height is N. The word PCF uses the array COUNTS to rapidly determine the level below which a given percentage of the area of the waveform is located.

Also, the word EDGE is defined, which refers to the array BEAM by the name BBEAM , to place markers outside the edges of the array BEAM to prevent a peak-finding program from going over the edge of BEAM and wandering into other areas of memory.

```
***    BLOCK NUMBER    525 DECIMAL    1015 OCTAL

  1  BASE @ DECIMAL
  2  : BLOCA THRESH = EDGE 0 QI = 2 0 DO BEGIN BEAM QI @ +
  3  @ THRESH @ < QI 1+= END QI @ QQI = BEGIN BEAM QI @ +
  4  @ BEAM QI @ + 1 - @ > QI @ 1 - QI = END QI @ 1 +
  5  QQI @ QI = BEGIN BEAM QI @ + @ THRESH @ > QI 1+= END
  6  QI @ QQI = BEGIN BEAM QI @ + @ BEAM QI @ + 1 + @ >
  7  QI 1+= END QI @ 1 -  QQI @ QI = LOOP ;
  8  : BIN GSET 0 LAWR 72 LRWR DLFR ;
  9  0 INTEGER M1 0 INTEGER M2 0 INTEGER P1 0 INTEGER P2
 10  : AMEA 0 ARE = TOTL DZ DO BEAM I + @ BOT - 0 MAX DUP I
 11  *D TOTL ++= ARE += LOOP ARE @ TOTL DDV ;
 12  : MNS !ST DUP !LE + SWAP AMEA M1 = !ST !LE + DUP !LE +
 13  SWAP AMEA M2 = ;
 14  : MGET BEAM M1 @ + @ P1 = BEAM M2 @ + @ P2 = ;
 15   OCTAL 7400 CONSTANT GCC DECIMAL
 16  BASE = ;S
```

This block defines the word AMEA , which provides the mean height of the beam in a fiven range, and the word BLOCA , which acts like the word PEAK in block 527, to find peaks in the waveform, here considered to be a beam profile.

```
***    BLOCK NUMBER    526 DECIMAL   1016 OCTAL

  1  BASE @ DECIMAL
  2  0 INTEGER DIF 0 INTEGER STR
  3  : VAPLO CINIT 0 CRWR 230 CVAL = 0 LAWR 72 LRWR DLFR
  4  DUP 2DEEP - DIF = STR = 24 0 DO 64 0 DO I 2 *
  5  DIF @ 128 */ BEAM + STR @ + @ GVAL 11 * I 2 * 1 +
  6  DIF @ 128 */ BEAM + STR @ + @ GVAL +
  7  384 + BACK @ + FGND @ + IMWR LOOP -10 CVAL += LOOP ;
  8  : XPP M2 @ !ST !LE - - 32 !LE */ 64 * 16 + CRWR
  9  32 0 DO BEAM !ST + I !LE 32 */ + @ 7 P1 @ */ 9 *
 10  GCC + IMWR LOOP
 11  M1 @ !ST - 32 !LE */ 16 + CRWR 32 0 DO BEAM !ST !LE
 12  + + I !LE 32 */ + @ 7 P2 @ */ 9 * GCC + IMWR CRRD
 13  63 + CRWR LOOP ;
 14
 15
 16  BASE = ;S
```

This block defines the word VAPLO , which, using the character set defined by the word GSET , draws part of the waveform in the array BEAM given the desired start and end channels in the form of a histogram with 128 points.

```
***    BLOCK NUMBER    527 DECIMAL   1017 OCTAL

  1  BASE @ DECIMAL
  2  : PEAK BEGIN BEAM QI @ + @ THRESH @ < QI 1+= END
  3  QI @ QQI = BEGIN BEAM QI @ + @ BEAM QI @ + 1 - @ >
  4  QI @ 1 - QI = END QI @ 1 + QQI @ QI =
  5  BEGIN BEAM QI @ + @ THRESH @ > QI 1+= END
  6 QI @ QQI = BEGIN BEAM QI @ + @
  7  BEAM QI @ + 1 + @ >
  8
  9
 10  QI 1+= END QI @ 1 -  QQI @ QI = ;
 11  : PREPAR SBM 80 PCF THRESH = EDGE 0 QI = ;
 12  : BSEAR 4 F = PEAK SWAP . BEGIN PEAK 2DEEP OVER OVER
 13  SWAP - 5 > IF SWAP . . ELSE DROP DROP THEN 500 QI
 14  @ > END . ;
 15  : TFIND 10 PCF 60 PCF DUP ' BOT = DUP ROT - + 3 +
 16  THRESH = ;                    BASE = ;S
```

This block defines the word PEAK , used for finding peaks in the waveform in BEAM , and the word PREPAR , which sets up the limits used normally as the criterion for peak finding. The exact limits of a peak as found by PEAK are defined as the last point of zero derivative preceding the start of a continuous area where the waveform is above the threshold value, and the first point of zero derivative following the end of that same continuous area.

```
***    BLOCK NUMBER    528 DECIMAL   1020 OCTAL

  1  BASE @ DECIMAL
  2
  3
  4
  5
  6
  7  : SDPL CRWR 182 CVAL = DUP 2DEEP - DIF = STR = 27 0 DO
  8  32 0 DO I 2 * DIF @ 64 */ BEAM + STR @ + @ GVAL 8 * I 2
  9  * 1 + DIF @ 64 */ BEAM + STR @ + @ GVAL +
 10  256 + BACK @ + FGND @ + IMWR LOOP -7 CVAL += CRRD 32 +
 11  CRWR LOOP ;
 12  : REPRO 0 LAWR 48 LRWR BEGIN LBLUE FGNDS VSAM 192 32
 13  0 SDPL YELLOW FGNDS HSAM 192 32 32 SDPL SS1 END ;
 14
 15
 16  BASE = ;S
```

This block defines the word SDPL , which draws a beam profile as does VAPLO , but only across half the screen, and with a given displacement from the left edge of the screen, and the word REPRO , which by calling SDPL can display both the horizontal and vertical parts of a beam profile on the screen at the same time, distinguished by color.

```
***    BLOCK NUMBER    529 DECIMAL   1021 OCTAL

 1  BASE @ DECIMAL
 2  CODE F5B S) LDA, 11 LGR, PUT
 3  OCTAL 3777 CONSTANT QMA 2000 CONSTANT DMAS DECIMAL
 4  0 INTEGER NDEP
 5
 6  : AWD BLARR IJ @ + @ IJ 1+= ;
 7  : ERRT DROP 8008 . DROP 31 ;
 8  : ER ' ERRT 4 - , ;
 9
10
11
12
13
14
15
16  BASE = ;S
```

This block contains a few basic words used for drawing diagrams on the screen, such as the word F5B , which returns the value of the first five bits of the word it operates on, which is where the "opcode" part of a word in the picture description language is.

```
***    BLOCK NUMBER    530 DECIMAL   1022 OCTAL

  1  BASE @ DECIMAL
  2  : FALS 1 NDEP = BEGIN AWD F5B DUP 16 EQ IF DROP NDEP
  3  1+= ELSE DUP 17 EQ IF DROP NDEP @ 1 EQ -1 * NDEP +=
  4  ELSE DUP 18 EQ IF DROP -1 NDEP += ELSE 31 EQ IF 0 ERRT
  5  THEN THEN THEN THEN NDEP @ END ;
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
 16  BASE = ;S
```

This block defines the word FALS . This word is invoked whenever there is an alternative in an IF clause in a picture description that is not taken. It reads succeeding IF, ELSE, and ENDIF directives in the picture description, keeping track of the nesting depth, and does not allow execution to resume until the end of the area not to be executed is reached.

The code operates as follows:

: FALS	
1 NDEP =	initialize the integer containing the nesting depth
BEGIN	begin the loop
AWD F5B	fetch next word of diagram description, and retain only the opcode

```
DUP 16 EQ                    making an extra copy for later use,
                             test to determine if the opcode
                             equals 16 (IF).

IF DROP NDEP 1+=             if so, discard the extra copy of
                             the opcode and increment the
                             nesting depth

ELSE DUP 17 EQ               otherwise, if the opcode is 17
                             (ELSE),

DROP NDEP @ 1 EQ -1 *        then, subtract one from the nesting
NDEP +=                      depth if and only if it equals 1;
                             in other words, ignore the ELSE
                             unless it belongs to the IF
                             instruction which caused FALS to be
                             executed in the first place

ELSE DUP 18 EQ               but if the opcode is 18 (ENDIF)

IF DROP -1 NDEP +=           always subtract 1 from the nesting
                             depth

ELSE 31 EQ                   also, check for end of diagram;

0 ERRT                       if encountered, indicate an error.

THEN THEN THEN THEN

NDEP @ END ;                 exit when the nesting depth becomes
                             zero
```

```
***    BLOCK NUMBER    531 DECIMAL   1023 OCTAL

  1  BASE @ DECIMAL
  2  : *CALL ERRT ; : *RTUN ERRT ; : *PROC ERRT ;
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
 16  BASE = ;S
```

This block defines the words used to interpret commands used for subroutines in picture descriptions; all these merely cause an error to be reported, as this is not yet implemented.

```
***    BLOCK NUMBER    533 DECIMAL   1025 OCTAL

  1  BASE @ DECIMAL
  2  : *ELSE DROP FALS ;
  3  : *IF QMA AND DUP DMAS AND IF STATUS @ AND ELSE
  4  STATUS @ NOT AND 0= THEN IF ELSE FALS THEN ;
  5  : *ENDIF DROP ; : *END DROP ;
  6  : *STOP DROP DROP 31 ;
  7  : *HCA COLR OR IMWR ;
  8  : *VCA QMA AND COLR OR IMWR CRRD 63 + CRWR ;
  9  : *POS QMA AND CRWR ;
 10  : *REL QMA AND CRRD + CRWR ;
 11  : *COLR QMA AND DUP 0= IF DROP GCOL @ ELSE THEN 512 *
 12  ' COLR = ;
 13
 14
 15
 16  BASE = ;S
```

This block defines words used to process various types of instructions in the picture description language.

Each of the words defined here, when called, expects the the picture description language instruction it is performing to be on the stack. Many of these words simply ignore that word, therefore removing it from the stack with the word DROP , but some of them use this part of the instruction for additional information.

The word ELSE , if encountered during execution rather than by FALS , indicates that the end of the true part of an IF clause has been reached: therefore, after using DROP to remove unneeded data from the stack, it calls the word FALS to skip the second part of the IF clause.

The word ENDIF , encountered during execution, is ignored, since execution, on leaving the IF clause, should then just continue normally.

The IF instruction has as its last 11 bits an indication of the condition to be tested: its first bit indicates whether all conditions mentioned should be true or just one, and each of the last ten bits indicates a condition stored bitwise in the variable STATUS .

The word *IF , which interprets the IF instruction, works as follows:

```
: *IF
QMA AND                    mask out the first five bits of the
                           word, leaving only the last 11 bits
                           of the instruction

DUP DMAS AND               making an extra copy of the last 11
                           bits, mask out all but the first of
                           these bits, leaving the bit that
                           indicates the type of test to be
                           performed

IF STATUS @ AND            if this bit is one, the contents of
                           the word STATUS and the last 10
                           bits of the instruction are ANDed
                           logically (no, the first bit of the
                           last 11 was not removed, but STATUS
                           is only 10 bits long) and the
                           result will be nonzero (and
                           therefore .TRUE.) if any 1 bits are
                           shared in common by STATUS and the
                           instruction

ELSE STATUS @ NOT AND      if the bit is zero, the last 10
0=                         bits are inverted and then an AND
                           with STATUS is performed; the
                           result will only be zero if every 1
                           bit in the instruction has a 1 bit
                           in STATUS corresponding to it; this
                           is the desired condition, so the
                           word 0= produces a 1 (.TRUE.) if
                           the result is zero, and a 0
                           (.FALSE.) otherwise.

THEN
```

IF ELSE FALS THEN ;	If the tested condition is true, fall through to normal execution in order to execute the first part of the IF clause; otherwise, call FALS to skip the part that is not to be executed.

The word *POS uses the last 11 bits of the instruction to provide a new position to the cursor; the word *REL adds the last 11 bits of the instruction to the cursor's current value.

The word *HCA writes the last 11 bits of the instruction to the screen as a character, after combining them with the current color in the word COLR ; the word *VCA does the same, but instead of allowing the cursor position to increment normally, it causes the cursor position to be increased by a total of 64 positions, thus causing successive characters to be written in a line going downwards rather than from left to right.

The word *COLR is used to interpret an instruction to change the current color; color zero, rather than being black, indicates that the color representing the gas currently in use in the ion source is to be used instead.

```
***    BLOCK NUMBER   534 DECIMAL   1026 OCTAL

  1  BASE @ DECIMAL
  2  ' *HCA 4 - INTEGER ADAR
  3  ' *VCA 4 - , ' *POS 4 - , ' *REL 4 - , ' *COLR 4 - ,
  4  ER ER ER ' *CALL 4 - , ER ER ER ER ER ' *RTUN 4 - ,
  5  ' *STOP 4 - ,
  6  ' *IF 4 - , ' *ELSE 4 - , ' *ENDIF 4 - , ER ER ER ER ER
  7  ' *PROC 4 - , ER ER ER ER ER ER ' *END 4 - ,
  8
  9
 10
 11
 12
 13
 14
 15
 16  BASE = ;S
```

This block sets up an array of FORTH definition addresses in the form required by the word SEL . As can be seen from the definition above, the following opcodes are defined at the present time:

```
 0 horizontal character
 1 vertical character
 2 reposition cursor
 3 displace cursor in relative co-ordinates
 4 select color

 8 call subroutine (unimplemented)
14 return from subroutine (unimplemented)
15 stop drawing diagram

16 if
17 else
18 endif

24 subroutine header (unimplemented)
31 end of diagram description
```

```
***    BLOCK NUMBER    535 DECIMAL   1027 OCTAL

  1  BASE @ DECIMAL
  2  : YDRAW 0 DCODE =
  3  31 LIM = 0 IJ = 1501 DIAGRAM @ 2 * - DUP
  4  BLARR BREAD 64 0 DO BLARR I + @ DARRA I 32 * + 31 + =
  5  LOOP 1 + BLARR BREAD
  6  DIAGRAM @ 1 EQ 0 LAWR IF 56 ELSE 57 THEN LRWR
  7  BEGIN BLARR IJ @ + @ DUP IJ 1+= F5B DUP 2DEEP
  8  ADAR SWAP SEL 31 - END ;
  9  1 CONSTANT ALL 2 CONSTANT TOP 3 CONSTANT MID
 10  4 CONSTANT MOB 5 CONSTANT CEN 6 CONSTANT MGP
 11
 12
 13
 14
 15
 16  BASE = ;S
```

This block defines the word YDRAW , which performs the act of drawing a diagram from a picture description stored in the array BLARR .

The word YDRAW operates as follows:

: YDRAW

0 DCODE =	the variable DCODE indicates the type of diagram that is on the screen, so that diagrams will not be erased unnecessarily
31 LIM =	the variable LIM is set as 32, as a safety precaution when executing SEL
0 IJ =	the variable IJ indicates which word of the picture description is currently in use

1501 DIAGRAM 2 * - DUP BLARR BREAD	the block whose number is determined by the formula 1501-(2*DIAGRAM) is read from the disk into BLARR , as it contains the screen positions of the names of the devices displayed (diagram numbers start with 1) Also, an extra copy of the blocknumber is left on the stack
64 0 DO BLARR I + DARRA I 32 * + 31 + = LOOP	This copies the screen positions of the names of the devices whose values are to be displayed in the current diagram into the device description block
1 + BLARR BREAD	the following disk block is read, as it contains the required diagram description
DIAGRAM @ 1 EQ	are we drawing diagram 1?
0 LAWR	before making this test, write 0 to the line address register,
IF 56 ELSE 57 THEN LRWR	then write either 56, if diagram 1, or 57, if any other diagram, to line file register zero: this selects 64-character lines for diagram 1, and 48-character lines for all other diagrams.
BEGIN	
BLARR IJ @ + @	fetch word IJ from the array BLARR ; in other words, the instruction in the picture description currently being executed,
DUP IJ 1+= F5B	make an extra copy, increment the "program counter", IJ , and from this copy remove the opcode part
DUP 2DEEP	make another copy of the opcode, this is used to check for the END instruction (31) but place it beneath the copy of the whole instruction word on the stack

```
ADAR SWAP SEL          execute the routine indicated by
                       the opcode, which must consume the
                       extra copy of the instruction word,

31 - END ;             then repeat the loop, unless a 31
                       is left on the stack... either an
                       END opcode or a fake one inserted
                       by the STOP instruction or the
                       error reporting routine
```

Also defined are abbreviations for the available diagrams in the form of constants: TOP for the diagram of the top end, and so on.

```
***    BLOCK NUMBER    537 DECIMAL   1031 OCTAL

  1  BASE @ OCTAL
  2  CODE /IRR S) LDA, BEGIN, 240 OTA, END,
  3  BEGIN, 1340 SKS, END,
  4  BEGIN, 1540 INA, END, S) STA,
  5  BEGIN, 1640 INA, END, PUSH
  6  : /IRD /IRR SWAP ;
  7  CODE !DSP S) LDA, SP IRS, IAB, S) LDA, SP IRS, IAB,
  8  S) DIV, S) STA, IAB, PUSH
  9  : DSP 2DEEP !DSP ;
 10  : CNSET 1 SWAP 0 DO 2 * LOOP 2 / OCIR ;
 11  DECIMAL  CODE L10S S) LDA, 10 LGL, PUT
 12  : BRANCH DUP IF 1 - L10S 8192 + THEN BRSC ;
 13  CODE LS1 S) LDA, 1 LGL, PUT
 14  : SHIFT 0 DO LS1 LOOP ;
 15
 16  BASE = ;S
```

This block defines the word /IRD , which allows a double-precision CAMAC read to be performed immediately, rather than as the result of a predefined CAMAC operation. Also defined are CNSET and BRANCH , which set crate and branch numbers on request, and the word SHIFT , which has the calling sequence:

w n SHIFT

and produces the result of w shifted n places left.

The word DSP is defined, with the calling sequence:

a b x DSP

and it divides the double-precision number (a,b) by x, returning the remainder at the top of the stack, and the quotient just below.

```
***    BLOCK NUMBER    538 DECIMAL   1032 OCTAL

 1  BASE @ DECIMAL
 2  : REP 0 DO DUP LOOP ;
 3  40 SECTOR
 4  4 ,CODE **/  0 , 0 , 0 , 0 ,
 5  ' **/ 1 + 2 / 2 *  8 REP
 6  SP LDX, 2 1) LDA, 4 1) MPY, CSA, 1 + STA,
 7  IAB, 2 + STA, CRA, SRC, AOA, STA, 2 1) LDA, 3 1) MPY,
 8  DBL, ADD, STA, SGL, 1 1) LDA, 4 1) MPY, DBL, ADD, STA,
 9  SGL, 1 1) LDA, 3 1) MPY, 15 LLS, DBL, ADD, SGL,
10  0 1) DIV, 3 1) STA, 2 + LDA, IAB, 0 1) DIV, 4 1) STA,
11  SP IRS, POP.
12   CODE */M SP LDX, 1 1) LDA, 2 1) MPY, 0 1) DIV,
13    1 1) STA, IAB, 2 1) STA, POP
14
15  4 INTEGER GCAR 3 , 5 , 2 , 7 ,
16  BASE = ;S
```

This block defines the array GCAR , which contains the colors used for the different gases that can be used in the ion source: red for hydrogen, light blue for deuterium, purple for Helium-3, green for Helium-4, and white for an unknown gas, or

Also, the word REP with the calling sequence:

x n REP

is defined, which places n extra copies of x on the stack (in addition to the original, which is not removed).

The words **/ and */M are defined in assembly language, for performing double-precision arithmetic. The word */M has the calling sequence:

a b c */M

and performs the arithmetic operation a*b/c, where a*b is allowed to be in double-precision form, returning the quotient at the top of the stack, and the remainder just

below.

The word **/ has the calling sequence

a b c d n **/

and returns as a double-precision number the product of (b,a) and (d,c), divided by n.

```
***    BLOCK NUMBER    539 DECIMAL    1033 OCTAL

  1  BASE @ DECIMAL
  2  59 SECTOR
  3  5 ,CODE *D// 0 , 0 , 0 , 0 , 0 ,
  4  ' *D// 1 + 2 / 2 * 16 REP
  5  SP LDX, 3 1) LDA, 5 1) MPY, CSA, 1 + STA, IAB, 3 + STA,
     CRA,
  6  SRC, AOA, STA, 3 1) LDA, 4 1) MPY, DBL, ADD, STA, SGL,
  7  2 1) LDA, 5 1) MPY, DBL, ADD, SGL, CSA, 1 + STA, IAB,
  8  2 + STA, CRA, SRC, AOA, STA, 2 1) LDA, 4 1) MPY, DBL,
     ADD,
  9  SGL, 1 1) DIV, STA, 2 + LDA, IAB, 1 1) DIV, 1 + STA,
     3 + LDA,
 10  IAB, 1 1) DIV, 1 + IMA, IAB, LDA, 0 1) DIV, 4 1) STA,
 11  1 + LDA, IAB, 0 1) DIV, 5 1) STA, SP IRS, SP IRS, POP.
 12  CODE DTC S) LDA, CMA, S) STA, SP LDX, 1 1) LDA,
 13  CMA, AOA, CSA, 1 1) STA, CRA, SSC, AOA, S) ADD, PUT
 14  : -- DTC ++ ;
 15  BASE = ;S
 16
```

This block defines the double precision subtraction operator, -- , and the operator *D// , which has the calling sequence:

m n a b *D//

and puts on the stack the result of the arithmetic operation ((m*n)/a)/b, where both m*n and m*n/a are allowed to be in double precision form.

```
***    BLOCK NUMBER    540 DECIMAL   1034 OCTAL

  1  BASE @ DECIMAL
  2  1 INTEGER QPA
  3  : ECALC DUP QPA = 10000 SWAP / 10 *D//
  4  OVER OVER 20000 1000 *D//
  5  5727 140 10000 1000 *D//
  6  OVER OVER OVER OVER 20000 10000 *D//
  7  49 0 ++ 1 0 100 **/ --
  8  1000 1000 *D//
  9  49 0 ++ 1 0 100 **/
 10  ;
 11
 12
 13
 14
 15
 16  BASE = ;S
```

This block defines the word ECALC , which calculates the beam energy from the NMR reading, assuming that the NMR reading is that of the analyzing magnet, that it is correct (which will not be true if the NMR magnetometer is in manual mode), and that the particle in use, and its charge, are known.

```
***    BLOCK NUMBER    541 DECIMAL   1035 OCTAL

  1  BASE @ DECIMAL
  2  0 CONSTANT NX ' NX CONSTANT XN  0 CONSTANT NZ
     ' NZ CONSTANT ZN
  3  0 CONSTANT NY ' NY CONSTANT YN  0 CONSTANT NS
     ' NS CONSTANT SN
  4  0 INTEGER DORDA 512 DP +=  DORDA 64 + CONSTANT DVALA
  5  DORDA 192 + CONSTANT DMESA
  6  : DLYX DELY 3 4 5 * * DROP ;
  7  : STEU BR4 1 OCIR NX 2 + @ WCO 7 0 DO DLYX LOOP ;
  8  : SDST NZ = ;
  9  : DST NZ DUP 2DEEP = 1 + = ;
 10  : *MUX STEU RCO SDST ;
 11  : *NMR STEU RCOD /CTOH DUP Q= IF DROP DROP ELSE DST
     THEN ;
 12  OCTAL  37777 CONSTANT CNMASK  DECIMAL
 13  : GADR NX 2 + @ ;
 14  : ADSET GADR DUP 4096 / BRANCH DUP 512 / 7 AND CNSET
 15  32 * CNMASK AND 0 FCNA ;
 16  BASE = ;S
```

This block defines several words belonging to the actual console scanner program itself. The variables NX , NY , NZ , and NS are defined to have both an INTEGER and a CONSTANT form, and are used to index four different tables for the entries they contain for the defice currently being scanned. Also, the word STEU is defined, which prepares for reading a device by setting up the branch and crate addresses, as well as executing a short delay required to ensure a successful read from the CAMAC interface.

The words SDST and DST are defined, which store one or two word results from devices in the array containing their most recent valid readings, and the words *MUX and *NMR are defined, which read either a standard multiplexer device or the nuclear magnetic resonance magnetometer respectively.

```
***    BLOCK NUMBER    543 DECIMAL   1037 OCTAL

 1  BASE @ DECIMAL
 2  DVALA 14 + CONSTANT MGV  DVALA 32 + CONSTANT MNR
 3  DVALA 12 + CONSTANT MGT  DVALA 16 + CONSTANT MTI
 4  : *ADC ADSET /IR DUP 2047 > IF 2048 SWAP - THEN SDST ;
 5
 6  : GASE GCAR + @ GCOL = ;
 7  : *GTY BR4 1 OCIR NX 2 + @ 4 0 DO DUP
 8  I + WC0 7 0 DO DLYX LOOP RC0 DUP 0= IF DROP ELSE
 9  MGV = I DUP MGT = GASE THEN LOOP DROP ;
10  : TST NZ DUP 2DEEP = DUP 2DEEP 1 + = 2 + = ;
11  0 14 0 /CNA 0 /R2 ATIM  0 14 1 /CNA 0 /R1 BTIM
12  : *CLK BR0 1 OCIR ATIM BTIM TST ;
13  : *PCA STEU RC0D /CTOH 100 DSP DUP 8 > IF DROP DROP
    ELSE
14  OVER 1000 / 1 > IF DROP DROP ELSE SWAP DST THEN THEN ;
15  : *SWR ADSET /IRD /CTOH DST ;   : *GVAL ;
16  BASE = ;S
```

This block defines four INTEGER type variables that refer to fixed locations in the table of parameter values so that the level of gas flow, the NMR reading, the gas type, and the current time may all be easily accessed; also, it defines the words *ADC , *GTY , *CLK , *PCA , and *SWR , which read, respectively, an A/D converter channel, the type of the gas in use, the current time, the picoammeter, and one of the switch registers. Actually, *ADC and *SWR are general single and double precision CAMAC read functions, respectively; their names only identify their first uses in the system.

```
***    BLOCK NUMBER    544 DECIMAL   1040 OCTAL

 1  BASE @ DECIMAL
 2  28 INTEGER RMAR  20754 ,  57 , 7809 ,  85 , 23581 ,
 3    113 , 25050 ,  85 , 23070 ,  113 , 24539 ,
 4  325 INTEGER RRAR  8449 ,  162 , 23254 ,  108 , 21218 ,
 5    81 , 28315 ,  108 , 21866 ,  81 , 28683 ,
 6  : AFET 2 * DUP RMAR + 1 + @ SWAP DUP RMAR + @ SWAP DUP
 7  RRAR + 1 + @ SWAP RRAR + @ MNR 1 + @ MNR @ ;
 8  : QST NZ DUP 2DEEP 2 + = DUP 2DEEP 3 + = DUP 2DEEP
 9  = 1 + = ;
10  : *ENE MGT @ DUP 4 EQ IF ELSE DUP AFET 7 FIND
11  2 < IF 1 ECALC DST DROP
12  ELSE 1 ECALC ROT 2 + AFET 2 ECALC QST THEN THEN ;
13
14
15
16  BASE = ;S
```

This block defines the arrays RMAR and RRAR , which contain the rest masses and the reciprocals of the rest masses of the six possible ions that are normally used in the accelerator: singly ionized hydrogen and deuterium, singly and doubly ionized Helium-3, and singly and doubly ionized Helium-4. Predefining the reciprocals as well as the rest masses themselves avoids attempting to obtain, within FORTH, the double-precision reciprocal of a double-precision number, thus saving some time.

Also defined in this block is the word QST , which is used like SDST and DST to return a value, but in this case a four word value, and the word *ENE , used to read the energy, which is a pseudo-device whose reading is calculated from the current NMR reading and the gas currently in use. If either isotope of helium is in use, the energies for both the singly and doubly ionized forms of the gas are

calculated. The word AFET is defined to place the rest mass and reciprocal rest mass of the ion in use on the stack in the form needed by the word ECALC .

```
***    BLOCK NUMBER    546 DECIMAL   1042 OCTAL

  1  BASE @ DECIMAL
  2  : CONV BEGIN BASE @ /MOD SWAP 3 FIND = SWAP 1 +
  3  SWAP DUP END DROP ;
  4  : DBLCONV 3 FIND 4 + 2DEEP 10000 DSP OVER IF
  5  4 FIND 4 0 DO DUP I + 0 SWAP = LOOP DROP
  6  2DEEP CONV 2DEEP CONV DROP ELSE
  7  2DEEP DROP DROP CONV THEN ;
  8  0 INTEGER DCVRA  11 DP +=
  9  : DCO DCVRA 2DEEP DBLCONV DCVRA - ;
 10  : 0> 0< 0= ;
 11  : SPFILL 12 0 DO -16 DCVRA I + = LOOP
 12  NX 21 + @ 1 MAX 0 DO 0 DCVRA I + = LOOP ;
 13  : SCO DCVRA SWAP DUP 0< IF ABS CONV DCVRA NX 21 + @ +
 14  MAX DUP -3 SWAP = 1 + DCVRA - ELSE CONV DCVRA - THEN ;
 15
 16  BASE = ;S
```

This block defines the words CONV and DBLCONV , used for converting numeric values into separate digits that may be subsequently displayed on the color display, and words SCO and DCO which call them, handling their calling sequences.

```
***    BLOCK NUMBER    547 DECIMAL   1043 OCTAL

  1  BASE @ DECIMAL
  2  0 INTEGER CKS
  3  : LETR CKS = CKS 2 * 1 + 1 TYPE ;
  4
  5
  6
  7  1 CONSTANT DEV
  8  : DVWR DEV IF OVER + IMWR ELSE DUP 32 < IF
  9  64 + THEN LETR THEN ;
 10
 11
 12
 13
 14  : ZFILL 12 0 DO 0 DCVRA I + = LOOP ;
 15
 16  BASE = ;S
```

This block defines the word LETR , which writes a character on the teletypewriter, and the word DVWR , which writes a character to either the color display or the teletypewriter, depending on the value of DEV . Also defined is the word ZFILL , which complements the word SPFILL , previously defined, by filling part of the output conversion buffer with zeroes rather than spaces.

```
***    BLOCK NUMBER    548 DECIMAL   1044 OCTAL

 1  BASE @ DECIMAL
 2  7 SECTOR HERE 63 , CODE CFET FETCH ANA, PUSH
 3  : Px NX 31 + @ 7 + CRWR ;
 4  : NAMER NX 22 + 2 * IP = NX 31 + @ CRWR
 5  NX 26 + @
 6  6 0 DO DUP CFET + IMWR LOOP DROP ;
 7  DCVRA 5 + CONSTANT DVRX
 8  : S# 6 0 DO DVRX I - @ 48 + DVWR
 9  I 5 NX 21 + @ - EQ IF 46 DVWR THEN LOOP DROP ;
10  DCVRA 8 + CONSTANT DVXR  : D# 9 0 DO DVXR I - @ 48 +
11  DVWR I 8 NX 21 + @ - EQ IF 46 DVWR THEN
12  LOOP DROP ;
13
14
15
16  BASE = ;S
```

This block defines the word Px , which positions the cursor on the color display where it should be to display the value read from the device currently being scanned by the console scanner, hereinafter reffered to as the current device.

Also, it defines the word S# , which displays a single precision number on the screen, and the word D# , which does the same for a double precision number, inserting an optional decimal point as well as a minus sign if required. As well, NAMER , used to write the short six-character name of a device on the screen, is defined here.

```
***    BLOCK NUMBER    549 DECIMAL   1045 OCTAL

  1  BASE @ DECIMAL
  2  : *ENERGY MGT @ 4 EQ IF ELSE NX 31 + @ 6 + DUP CRWR
  3  SPFILL NZ DUP @ SWAP DUP 1 + @ ROT DCO DROP
  4  NX 26 + @ DUP D# SWAP SPFILL
  5  MGT @ 2 * 7 - ABS 2 < IF
  6  ROT 11 + CRWR DUP 3 + @ SWAP 2 + @ DCO DROP
  7  D# ELSE DROP DROP DROP THEN THEN ;
  8
  9
 10  : *PICO NX 31 + @ 6 + DUP CRWR SPFILL
 11  NZ DUP @ SCO DROP NX 26 + @ DUP S# SWAP
 12  SPFILL ROT 6 + CRWR 1 + @ SCO DROP S# ;
 13
 14
 15
 16  BASE = ;S
```

This block defines the words *ENERGY and *PICO , which are used to *display* the values of the energy and of the picoammeter, respectively.

```
***    BLOCK NUMBER    550 DECIMAL    1046 OCTAL

 1  BASE @ DECIMAL
 2  : &DPI SPFILL NZ DUP 1 + @ SWAP @
 3  DCO DROP NX 26 + @ D# ;
 4  : &SPI SPFILL NZ @ NX 20 + @ *
 5  SCO DROP NX 26 + @ S# ;
 6
 7  : V# DO DVRX I - @ 48 + DVWR LOOP DROP ;
 8  : &TIME BASE @ HEXADECIMAL SPFILL NZ DUP
 9  1 + @ SCO DROP NX 26 + @ DUP S# 58 DVWR ZFILL
10  OVER 2 + @ SCO DROP DUP 4 2 V# 58 DVWR DUP 6 4 V#
11  46 DVWR SPFILL SWAP @ SCO DROP DUP 6 5 V#
12  32 DVWR 5 2 V# BASE = ;
13  : *TIME NX 31 + @ 6 + CRWR &TIME ;
14  : *DPI NX 31 + @ 6 + CRWR &DPI ;  : *SPI Px &SPI ;
15
16  BASE = ;S
```

This block defines the words *TIME , *SPI , and *DPI , which are used to display the time, a single precision value, and a double precision value respectively, as well as the word V# , used to display pieces of the output conversion buffer, and &TIME , &SPI , and &DPI , which omit cursor positioning and thus are used in the header portion of printed logouts of beam conditions, where display code cannot serve a dual purpose.

```
***    BLOCK NUMBER   551 DECIMAL    1047 OCTAL

  1  BASE @ DECIMAL
  2
  3
  4
  5
  6  ' *MUX 4 - INTEGER SADAR
  7  ' *SWR 4 - , ' *NMR 4 - , ' *ADC 4 - , ' *PCA 4 - ,
  8  ' *CLK 4 - , ' *ENE 4 - ,
  9  ' *GTY 4 - , ' *GVAL 4 - ,
 10
 11
 12
 13
 14  : RESTO BR4 1 OCIR ;
 15  : QSCAN SADAR SWAP SEL RESTO ;
 16  BASE = ;S
```

This block defines the word QSCAN , which uses the array SADAR , containing the addresses of the various device read routines, to read the current device given its general type as a number from zero to eight.

```
***    BLOCK NUMBER    552 DECIMAL   1050 OCTAL

  1  BASE @ OCTAL  10 SECTOR HERE DUP DUP 1 + 7000 , DECIMAL
  2  63 , CODE CACQ FETCH ANA, ADD, PUSH  CONSTANT COLOR
  3  : STYPE SWAP IP = 0 DO CACQ IMWR LOOP ;
  4  : SAY COUNT STYPE ;
  5  : &ENERGY MGT @ 4 EQ IF ELSE SPFILL NZ DUP @ SWAP DUP
     1 + @
  6  ROT DCO DROP 0 D# SPFILL MGT @ 2 * 7 - ABS 2 < IF DUP
  7  3 + @ SWAP 2 + @ DCO DROP 0 D# ELSE DROP THEN THEN ;
  8  : &PICO NZ DUP ? 1 + ? ;
  9  OCTAL 74000 INTEGER GCLS 31000 , 65000 , 20000 ,
 10  16000 , DECIMAL
 11  #  P     D     HE3   HE4   NONE  # GNAMES
 12  GNAMES 1 + CONSTANT GNAMES
 13  : NDIS 2 * IP = Px 6 0 DO DUP CFET 63 AND +
 14  IMWR LOOP DROP ;
 15  : *GTYPE NZ @ DUP GCLS + @ SWAP 3 * GNAMES + NDIS ;
 16  BASE = ;S
```

This block defines the words &ENERGY and &PICO , which, because they print more than one number, have to be redefined in full for printing purposes, the word SAY , which types a message on the screen, and the word *GTYPE , which displays the type of gas in use by printing its name on the screen (or the teletypewriter).

```
***    BLOCK NUMBER    553 DECIMAL   1051 OCTAL

  1  BASE @ DECIMAL
  2  0 INTEGER CLO  0 INTEGER TCR  0 CONSTANT UVR
  3  2 CONSTANT NH
  4  : CLL NZ 1 + @ SS4 0= IF DUP CLO @ XOR NH AND IF
  5  1 % UVR = THEN THEN CLO = ;
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
 16  BASE = ;S
```

This block defines the word CLL , which causes a printed logout to take place every two hours (changing the value of the constant NH to another power of two can change this) if sense switch four is set. This is the only control action currently defined in the system.

```
***    BLOCK NUMBER    554 DECIMAL   1052 OCTAL

  1  BASE @ DECIMAL
  2  ' DROP 4 - INTEGER CADAR
  3  ' CLL 4 - ,
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15  : CONTROL 1 LIM = CADAR SWAP SEL ;
 16  BASE = ;S
```

This block defines the word CONTROL , which selects the appropriate control action for the current device and executes it.

```
***    BLOCK NUMBER    557 DECIMAL   1055 OCTAL
 1  BASE @ DECIMAL
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15  : ALARM DROP ;
16  BASE = ;S
```

This block defines the word ALARM , which selects the appropriate user signalling action for the current device and executes it. At present, it is a dummy routine.

```
***    BLOCK NUMBER    558 DECIMAL   1056 OCTAL

  1  BASE @ DECIMAL
  2  : TCTRL DUP 2 AND IF NX 3 + @ ALARM THEN
  3  1 AND IF NX 4 + @ CONTROL THEN ;
  4
  5
  6
  7
  8  ' *SPI 4 - INTEGER DADAR
  9  ' *DPI 4 - , ' *PICO 4 - , ' *TIME 4 - ,
 10  ' *ENERGY 4 - , ' *GTYPE 4 - ,
 11
 12  : DISPLAY DADAR NX 5 + @ SEL ;
 13
 14
 15  0 INTEGER XJ
 16  BASE = ;S
```

This block defines the word TCTRL , which tests to see if either control or alarm actions are required, and if so calls one or both of the words ALARM and CONTROL . Also, the word DISPLAY is defined, which selects the appropriate display routine from the list DADAR and calls that routine to display the value read from the current device.

```
***    BLOCK NUMBER    558 DECIMAL   1056 OCTAL

  1  BASE @ DECIMAL
  2  0 INTEGER JAR 0 , 0 ,
  3
  4
  5
  6  : &GTYPE NZ @ 3 * GNAMES + 2 * 6 TYPE ;
  7  BEAM CONSTANT LARA
  8  : LNPR LARA XJ @ + 1 + 2 * 16 TYPE BL ;
  9  : UPR LARA XJ @ + 9 + 2 * 6 TYPE  ;
 10  : NPR NX 22 + 2 * 6 TYPE BL ;
 11  ' &SPI 4 - INTEGER PADAR
 12  ' &DPI 4 - , ' &PICO 4 - , ' &TIME 4 - ,
 13  ' &ENERGY 4 - , ' &GTYPE 4 - ,
 14
 15  : PNUM PADAR NX 5 + @ SEL ;
 16  BASE = ;S
```

This block defines the word &GTYPE , which prints the current gas type, and the word PNUM , which is the teletypewriter analog of DISPLAY .

```
***    BLOCK NUMBER    562 DECIMAL   1062 OCTAL

  1  BASE @ DECIMAL  0 INTEGER NFLAG
  2  : SCAN 0 SN =
  3  BEGIN DORDA NS + @ DUP 32 * DARRA + XN = DUP
  4  5 * DMESA + YN = 2 * DVALA + ZN = NX DUP @ DUP 0<
  5   IF 32767 AND OVER 31 + @ 0> DCODE @ 0= *
  6    IF 1 1
  7    ELSE DUP 7 AND
  8     IF 0 1 ELSE 0 THEN THEN
  9    IF ROT 1 + @ 8 LIM = QSCAN
 10     IF 5 LIM =
 11      DISPLAY THEN
 12     TCTRL 1
 13    ELSE DROP DROP 1 THEN
 14   ELSE SWAP DROP 8 - THEN 1 SN +=
 15  END ;
 16  BASE = ;S
```

This block defines the word SCAN . This word is the inner loop of the console scanner; it is this word that steps through all the devices in order, each one becoming the current device in turn, first fetching the information about the current device from the appropriate tables, then reading the device, and subsequently calling the routines to display its value, check for alarm or control actions to be done, if such actions are requested by the device description block entry for that device.

The word SCAN functions as follows:

```
: SCAN
```

```
0 SN =
```

start from the beginning of the table giving the device description table position of the devices to be scanned, in the order of scanning

BEGIN	outermost loop
DORDA NS +	Get the device description table position of the current device from the array DORDA
DUP 32 * DARRA + XN =	set up XN (CONSTANT form NX) to point to the start of the device description table entry for the current device
DUP 5 * DMESA + YN =	set up YN (or NY) to point to the area of the message array DMESA used for the activity flag, the device pre-emption flag, the priority number, the mailbox word, and the future device pre-emption flag associated with the current device
2 * DVALA + ZN =	set up ZN (or NZ) to point to the two words of the array DVALA used for the value read from the current device. (The device ENERGY, which has a four-word value, has a position adjacent to a device with no value to place in this array.)
NX DUP @ DUP 0<	Now on the stack: a one if the first bit of the current device's device description table is set, and a zero otherwise, in which case the device is not scanned; a copy of the first word of the device description table entry belonging to the current device; and a pointer to the device description table entry of the current device
IF	if the first number mentioned above is 1, proceed to scan the current device
32767 AND	strip the first bit from the copy of the first word of the current device's device description table entry

OVER 31 + @ 0>	get a copy of the pointer to the current device's device description table entry, and use it to fetch a copy of word 31 of that entry; this memory location, set up by the word YDRAW , defined in block 535, contains either the cursor location for writing the name of the current device on the screen, or -1 if the device is not to be displayed; thus, the word 0> is used to get a logical flag indicating whether or not to attempt display of the current device
DCODE @ 0= *	verify that DCODE is zero: if not, the color display is being used for a different purpose, and no devices are to have their readings displayed
IF	is the current device to be displayed?
1 1	place two ones on the stack to indicate this to the next part of the program if so
ELSE	otherwise
DUP 7 AND	examine the last three bits of the first word of the device description table entry for the current device: these bits indicate if this device is to be scanned even if its value is not displayed and no control and alarm actions are initiated by it, if alarm actions are to be taken in response to readings from this device, and if control actions are to be taken in response to readings from this device. In each case, a 1 bit indicates that the action is to be performed.
IF	if any of these bits are nonzero,
0 1	indicate that scanning, but not display, is to take place

```
    ELSE                    otherwise
    0                       indicate that scanning is not to
                            take place
   THEN
  THEN
 IF                         is scanning to take place?
  ROT 1 + @                 if so, take the pointer to the
                            current device description table
                            entry, and use it to obtain a copy
                            of word 1 of that entry (its second
                            word)
  8 LIM = QSCAN             that word contained the general
                            type of the device for the purpose
                            of reading its value; thus, QSCAN
                            is used to perform the read
  IF                        is the value to be displayed?
   DISPLAY                  display it
  THEN
  TCTRL 1                   test for and perform control; then
                            leave a 1 on the stack to indicate
                            that scanning is to continue
  ELSE                      if display does not take place
  DROP DROP 1               discard unneeded data
 THEN
 ELSE                       if scanning does not take place
 SWAP DROP 8-               if word zero of the device
                            description table entry associatted
                            with this device is equal to 8,
                            this is a dummy device used to
                            indicate the end of the list
THEN
1 SN +=                     step to next device
```

```
END ;                    repeat loop unless last device,
                         then exit
```

```
***    BLOCK NUMBER    564 DECIMAL  1064 OCTAL

 1  BASE @ DECIMAL
 2  : DLL 1000 0 DO DLYX LOOP ;
 3  0 CONSTAND UVX
 4  : TINT EXSTL ?QV
 5  IF DLL 0 EXCL
 6  ELSE UVR
 7   IF 1 ' UVX = 0
 8   ELSE 1
 9   THEN
10  THEN ;
11
12
13
14
15
16  BASE = ;S
```

This block defines the word TINT , which tests for a Look-at-Me signal from the color display controller resulting from the user pushing the red button attatched to its external LAM input.

```
***    BLOCK NUMBER    565 DECIMAL   1065 OCTAL

 1  BASE @ DECIMAL
 2
 3  OCTAL 7000 INTEGER ALCAR 6000 , 2000 , 5000 ,
 4  3000 , DECIMAL
 5  0 INTEGER CPTR  : RTYPE ALCAR CPTR @ + @ COLOR =
 6  STYPE CPTR 1+= CPTR @ 4 > IF 0 CPTR = THEN ;
 7  : TVSL 0 LAWR 65 LRWR SERS 0 CPTR =
 8  BLK = 17 1 DO I LINE I 1 - 128 *
 9  2 0 DO OVER OVER CRWR 2 * 16 RTYPE
10  OVER OVER 24 + CRWR 8 + 2 * 16 RTYPE
11  64 + SWAP 16 + SWAP LOOP DROP DROP LOOP
12  ECUR EEXTCUR ;
13  : MVH MINUS MOVE ;
14  : CRD BLARR BREAD BLARR DVALA 128 MVH ;
15  : CSV DVALA BLARR 128 MVH BLARR BWRITE ;
16  BASE = ;S
```

This block defines the array ALCAR , containing the colors used for display of the various gas names, a block move subroutine, MVH , with a calling sequence improved over that of the one provided in the system, and the words CRD and CSV , which read and save machine conditions on the disk.

Also defined is the word TVSL , which is used in displaying menus of alternatives on the screen, from which the user makes a choice by moving the cursor and then pushing the red External LAM Input button connected to the color display controller.

```
***    BLOCK NUMBER    567 DECIMAL   1067 OCTAL

  1  BASE @ DECIMAL
  2  # DIAGRAM?# DM1
  3
  4  # BLOCK NUMBER?# DM3
  5  0 INTEGER MDP
  6  # WHAT YEAR IS THIS?# YM1
  7  # GAS TO BE USED:(P:1/D:2/HE3:3/HE4:4)# GM1
  8  # BELT TIME# TM1
  9  # PROBE TIME# TM2
 10  # TUBE TIME# TM3
 11  # RF DEFLECTION TIME# TM4
 12  # ____________________________# UM1
 13  # PROFILE:HORIZ(0),VERT(1),OR FID(2)# FM1
 14  # TO/FROM CHANNELS FOR PLOT# FM2
 15
 16  BASE = ;S
```

This block defines a number of messages printed by the computer, either in dialog with the user or in printed logouts.

```
***    BLOCK NUMBER    568 DECIMAL   1070 OCTAL

  1  BASE @ DECIMAL
  2  # RADIATION LEVELS  (MILLIRADS/HOUR)# RM1
  3  # OLD TARGET ROOM  CONTROL ROOM     STUDE
     NT STUDY  FENCE# RM2
  4  # NEUTRONS# RM3
  5  # GAMMAS# RM4
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
 16  BASE = ;S
```

This block defines more messages used in printed logouts.

```
***    BLOCK NUMBER    569 DECIMAL   1071 OCTAL

  1  BASE @ DECIMAL
  2  : %CONSA DM3 COUNT TYPE
  3  BEGIN BEGIN ?? DUP 2000 < END DUP 5000 > END CSV 1 ;
  4  : %GAS MDP @ @ GASE 1 ;
  5  : %DIAGRAM MDP @ @ DIAGRAM = 1 ;
  6  : %STOP 0 0 ;
  7
  8  : %PROPLO 2 LIM = FM1 COUNT TYPE ?? 301 LOAD
  9  2 DCODE = 1 ;
 10
 11
 12
 13
 14
 15  BASE = ;S
 16
```

This block defines words that perform actions such as respecifying the diagram to display, or the gas in use, in response to user requests through the menu.

```
***    BLOCK NUMBER    571 DECIMAL   1073 OCTAL

 1  BASE @ DECIMAL
 2  0 CONSTANT YC
 3  : MOCOMP 16 / DUP 16 / 6 * - DUP 160 / 60 * -
 4  59 - YC - DUP 1 < IF YC + 365 + THEN
 5  DUP 2 * 1 - 5 * 306 / SWAP OVER 4 * 2 - 153 * 5 /
 6  63 + 4 / - SWAP ;
 7  #  MAR APR MAY JUNEJULYAUG SEPTOCT NOV DEC JAN FEB #
    MARA
 8  MARA 1 + 2 * CONSTANT MARA
 9  : GREG DUP 100 MOD 0= IF 100 / THEN 4 MOD 0= ;
10
11  : DATE MTI @ MOCOMP 3 F = 4 * MARA + 4 TYPE . ;
12  : DAY MTI 3 + @ DUP GREG ' YC = DATE 5 F = . ;
13
14
15
16  BASE = ;S
```

This block defines words used for converting day numbers for a year into dates of the form month/day/year, given the year.

```
***    BLOCK NUMBER    573 DECIMAL    1075 OCTAL

 1  BASE @ DECIMAL
 2  0 INTEGER CP  #                  # NSPA
 3  : SETAB 3 0 DO NSPA COUNT TYPE LOOP 155 LETR 177 LETR
 4  NSPA COUNT TYPE 155 LETR 177 LETR ;
 5  : TAB 137 LETR DLL ;
 6  : LSP CP @ IF TAB ELSE CR THEN 1 CP @ - CP = ;
 7  : USC CR CR UM1 COUNT TYPE TAB UM1 COUNT TYPE
 8  CR CR CR;
 9
10
11
12
13
14
15
16  BASE = ;S
```

This block contains words that use the tabulator feature of the Anderson-Jacobson 860 teletypewriter terminal to speed up production of printed logouts by setting up the required tabulator stops.

```
***    BLOCK NUMBER    574 DECIMAL   1076 OCTAL

  1  BASE @ DECIMAL
  2  : LOGOU CR SETAB CR
  3  0 ' DEV = DAY CR MTI DUP ZN = DVALA - 16 * DARRA + XN =
  4  &TIME CR CR
  5  TM1 COUNT TYPE TAB TM2 COUNT TYPE USC
  6  TM3 COUNT TYPE TAB TM4 COUNT TYPE USC
  7  7 F = 0 XJ = 1 CP =  BEGIN LARA XJ @ + @ DUP
  8  2 * DVALA + ZN = DUP 32 * DARRA + XN =
  9  0 DMESA = 0< IF LARA XJ @ + @ -1 * 1 - DUP 0= 0=
 10  IF LSP THEN ELSE LNPR 5 LIM = PNUM BL UPR LSP 1
 11  THEN 12 XJ += END 1 ' DEV = CR CR CR RM1 COUNT TYPE
 12  CR CR NSPA COUNT TYPE RM2 COUNT TYPE CR CR
 13  RM3 COUNT TYPE CR CR RM4 COUNT TYPE 17 0 DO CR LOOP ;
 14
 15
 16  BASE = ;S
```

This block defines the word LOGOU , which acts in a manner somewhat analogous to that of the word SCAN to print out a record of machine conditions.

: LOGOU	
CR SETAB CR	return the carriage, set the teletypewriter tabulator stops, and return the carriage again to begin printing
0 ' DEV =	indicate that output is directed to the teletypewriter and not the color display
DAY	print the date
MTI DUP ZN = DVALA - 16 * DARRA + XN =	set both value storage and device description block pointers for the clock
&TIME CR CR	print the time of day, leave a blank line

```
TM1 COUNT TYPE TAB
TM2 COUNT TYPE USC
TM3 COUNT TYPE TAB
TM4 COUNT TYPE USC
```

print spaces, appropriately labeled, for the operator to write the belt time, tube time, RF deflection time, and probe time from the cumulative time meters in the console

```
7 F =
```

use seven spaces, normally, for directly printed numbers, as opposed to numbers printed through DVWR .

```
0 XJ =
```

start with the first device

```
1 CP =
```

on the left-hand column of the page

```
BEGIN
```

```
LARA XJ @ + @
```

get the element of the logout array corresponding to the current device

```
DUP 2 * DVALA + ZN =
```

set up the value storage pointer

```
DUP 32 * DARRA + XN =
```

and the device description table pointer

```
0 DMESA =
```

clear message from device zero, the operator

```
0< IF
```

if the address of the current device is negative,

```
LARA XJ @ + @ -1 * 1 -
DUP 0= 0=
```

make it positive, and subtract one, so as to obtain a zero for -1, but a one for -2; make sure that the result is indeed 0 or 1 for the top copy; on -1, a zero is placed on the stack to indicate the end of the logout; on -2,

```
IF LSP THEN
```

space one field before printing the value from the next device in the sequence

```
ELSE
```

if the address is positive,

```
LNPR 5 LIM = PNUM
```

print the name of the device, and its reading

```
BL UPR LSP 1
```
space to the next printing position, and indicate that a device has been read

```
THEN
```

```
12 XJ +=
```
proceed to the next device

```
END
```
repeat loop, unless end of list is reached

```
1 ' DEV =
```
restore the device type indicator

```
CR CR CR RM1 COUNT
TYPE CR CR NSPA COUNT
TYPE RM2 COUNT TYPE CR
CR RM3 COUNT TYPE CR
CR RM4 COUNT TYPE
```
type more messages, appropriately spaced, to provide an area in which the operator may write down the current radiation levels in various areas,

```
17 0 DO CR LOOP ;
```
finally, provide enough space so that the whole thing fits neatly on one page between the perforations in the paper

```
***    BLOCK NUMBER    575 DECIMAL   1077 OCTAL

 1  BASE @ DECIMAL
 2  : %LOGO 1993 LARA BREAD 1994 LARA 512 + BREAD
 3  LOGOU 1 ;
 4  ' %GAS 4 - INTEGER MRADAR
 5  ' %DIAGRAM 4 - , ' %STOP 4 - , ' %CONSA 4 - ,
 6  ' %LOGO 4 - , ' %PROPLO 4 - ,
 7
 8
 9
10  : CMENU 5 LIM = MRADAR SWAP SEL ;
11
12
13
14
15
16  BASE = ;S
```

This block defines the word CMENU , which handles response to user responses to the menu displayed on the color screen.

```
***    BLOCK NUMBER    577 DECIMAL   1101 OCTAL

  1  BASE @ DECIMAL
  2  : SINT DCODE @ DUP 1 EQ IF UVX IF 1 ELSE
  3  DROP DEXTCUR DCUR
  4  1991 BLARR BREAD
  5  CRRD DUP 64 / SWAP 63 AND 23 > 27 * + BLARR + DUP
  6  256 + MDP = @ CMENU DCODE @ 2 EQ IF ELSE SERS 0 DCODE =
  7  0 THEN THEN ELSE 2 EQ IF UVX IF 1 ELSE 1 0 THEN
  8  ELSE UVX 0=
  9  IF SS1 IF SS2 IF SS3 IF
 10  1 DCODE = 1990 TVSL 23 CRWR ECUR EEXTCUR 1
 11  ELSE LOGOU 1 THEN
 12  ELSE DM1 COUNT TYPE ?? DIAGRAM = 1 0 THEN
 13  ELSE 0 0 0 THEN
 14  ELSE TCR @ 0= IF 0 ' UVR = LOGOU THEN 1 THEN
 15  THEN THEN 0 ' UVX = ;
 16  BASE = ;S
```

This block defines the word SINT , which selects and calls the subroutine required to perform the action the user has requested from the menu displayed on the color monitor. It leaves 1, 0 and then 1, two zeroes and a one, or three zeroes on the stack on exit so as to allow different branches back to earlier parts of the word SCANNER , which calls it.

```
***    BLOCK NUMBER    581 DECIMAL    1105 OCTAL

 1  BASE @ DECIMAL
 2
 3
 4
 5  : SCANNER BEGIN 0 NFLAG = 4 MGT = 2 DCODE = SCAN
 6  MGT @ 4 EQ IF GM1 COUNT TYPE ?? 1 - MGT = THEN
 7  BEGIN SERS 300 LOAD
 8
 9  1993 LARA BREAD 1994 LARA 512 + BREAD 1 NFLAG =
10
11  BEGIN BEGIN SCAN 0 NFLAG = TINT END SINT END END END ;
12  : ALLSCAN BEGIN 6 1 DO I DIAGRAM = SERS SCANNER LOOP
13  SS1 END ;
14
15
16  BASE = ;S
```

This block defines the word SCANNER , which is invoked as the main program of the console scanner. Also defined is the word ALLSCAN , which calls SCANNER in such a way as to permit rapid cycling through the different screen formats available.

The word SCANNER works as follows:

: SCANNER	
BEGIN	it must be possible for the user to return the program to this point without stopping the computer, but by issuing a request within the normal operation of SCANNER
0 NFLAG =	formerly, this variable was used used to have SCAN write device names on the screen: this function is now performed by YDRAW
4 MGT =	gas type is not known yet

```
2 DCODE =                    one scan is performed with a blank
                             screen, to determine the type of
                             gas in use

SCAN

MGT @ 4 EQ                   is the gas type still not known (if
                             the amount of gas flow is zero,
                             this will happen, due to a
                             peculiarity of the interface used)?

IF                           if so,

GM1 COUNT TYPE ?? 1 -        ask the user to identify the gas
MGT =                        being used, and use the answer
                             given

THEN

BEGIN                        another branch back point

SERS 300 LOAD                clear the screen and draw the
                             required diagram

1993 LARA BREAD 1994         read in the logout information from
LARA 512 + BREAD             the disk, so that it can be
                             resident during normal operation

1 NFLAG =                    not used

BEGIN BEGIN                  two different ways to return here

SCAN                         perform a scan of all the devices
                             once

0 NFLAG =                    not used

TINT                         check for user service request.

END                          if no request, a 1 is placed on the
                             stack; otherwise, a 0 is so placed,
                             and we fall through

SINT                         to service that request,

END END END ;                and depending on the type of the
                             request, we may resume scanning at
                             various points in the routine
                             SCANNER , depending on how much is
                             to be changed.
```

```
***    BLOCK NUMBER    584 DECIMAL   1110 OCTAL

  1  BASE @ DECIMAL
  2  : WTUA 4 0 DO 1999 I - DARRA 512 3 I - * + BWRITE
  3  LOOP ;
  4  0 INTEGER DWOR  : CURR 32 * DARRA DWOR @ + + ? ;
  5  : RPL SWAP 32 * DARRA DWOR @ + + = ;
  6  : ENQ 32 * DARRA 22 + + 2 * 6 TYPE ;
  7  : NRE ILIN 32 * DARRA 22 + + 3 0 DO
  8  DUP I + ITEX I 2 * + DUP 1 + @ SWAP @
  9  256 * + SWAP = LOOP DROP ;
 10
 11
 12
 13
 14
 15
 16  BASE = ;S
```

This block defines the word WTUA , which writes the device description block back out on the disk (where it occupies the four blocks of storage from 1996 to 1999 inclusive), after it has been edited. Also defined are the words RPL , ENQ , CURR , and NRE , which were described with block 450, where they are also defined.

```
***    BLOCK NUMBER    586 DECIMAL   1112 OCTAL

 1  BASE @ DECIMAL
 2  : LENQ 12 * LARA 1 + + 2 * 16 TYPE ;
 3  : UEN ILIN 12 * LARA 9 + + 3 0 DO
 4  DUP I + ITEX I 2 * + DUP 1 + @ SWAP @ 256 * + SWAP =
 5  LOOP DROP ;
 6
 7
 8
 9
10
11
12
13
14
15
16  BASE = ;S
```

This block is a duplicate of block 452.

```
***    BLOCK NUMBER   588 DECIMAL   1114 OCTAL
  1  BASE @ DECIMAL
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11  : STUA 4 0 DO 1996 I + DARRA 512 I * + BREAD LOOP ;
 12
 13
 14
 15
 16  BASE = ;S
```

This block defines the word STUA , which reads the device description block from its location on the disk into memory, for use by the console scanner.

```
***    BLOCK NUMBER    590 DECIMAL   1116 OCTAL

 1  BASE @ DECIMAL
 2  : PDISL DIAGRAM = 300 LOAD DARRA XN = 0 YN = DVALA ZN =
 3  BEGIN NX @ DUP 0<
 4  IF DROP NX 31 + @ 0> IF NAMER 5 LIM = DISPLAY THEN 1
 5  ELSE 8 - THEN 2 ZN += 5 YN += 32 XN += END ;
 6
 7
 8
 9
10
11
12
13  : AINIT CINIT SERS DLFR DGCS GSET STUA
14  1995 DORDA BREAD
15  YM1 COUNT TYPE ?? MTI 3 + = ;
16  BASE = ;S
```

This block defines the word PDISL , which displays on the screen a set of machine conditions previously read from the disk in the same form as used during the operation of the console scanner; also defined is the word AINIT , which takes care of all initialization that is required before running the console scanner.